Henning Bordihn, Géza Horváth, György Vaszil (eds.)

Twelfth Workshop on Non-Classical Models of Automata and Applications (NCMA 2022)

August 26 – 27, 2022, Debrecen, Hungary

Short Papers

UNIVERSITY *of* DEBRECEN

NCMA 2022, Twelfth Workshop on Non-Classical Models of Automata and Applications, Debrecen, Hungary, 2022, Short Papers [PDF]

Published by the Faculty of Informatics of the University of Debrecen, Kassai út 26, 4028 Debrecen

Edited by © Henning Bordihn, Géza Horváth, György Vaszil, 2022

Copyright © Authors of the contributions, 2022

Published in August 2022

Preface

This volume contains the five short contributions of the *Twelfth Workshop on Non-Classical Models of Automata and Applications (NCMA 2022)* held in Debrecen, Hungary, on August 26 and 27, 2022.

The NCMA workshop series was established in 2009 as an annual event for researchers working on non-classical and classical models of automata, grammars or related devices. Such models are investigated both as theoretical models and as formal models for applications from different points of view. The goal of the NCMA workshop series is to exchange and develop novel ideas in order to gain deeper and interdisciplinary coverage of this particular area that may foster new insights and substantial progress.

The previous NCMA workshops took place in the following places: Wrocław, Poland (2009), Jena, Germany (2010), Milano, Italy (2011), Fribourg, Switzerland (2012), Umeå, Sweden (2013), Kassel, Germany (2014), Porto, Portugal (2015), Debrecen, Hungary (2016), Prague, Czech Republic (2017), Košice, Slovakia (2018), Valencia, Spain (2019). Due to the Covid-19 pandemic there was no NCMA workshop in 2020 and 2021. The Twelfth Workshop on Non-Classical Models of Automata and Applications (NCMA 2022) was organized by the Faculty of Informatics of the University of Debrecen. It was co-located with DCFS 2022, the 24th International Conference of Descriptional Complexity of Formal Systems (August 29 - 31) and MCU 2022, the 9th Conference on Machines, Computations and Universality (August 31 - September 2, 2022).

Since 2014, in addition to the invited talks and regular contributions, the NCMA workshops have also included short presentations reporting on recent results or ongoing work. Nevertheless, each short presentation has been evaluated by members of the Program Committee. This volume contains the extended abstracts of the five short papers selected for presentation at the Twelfth Workshop on Non-Classical Models of Automata and Applications in Debrecen.

We are grateful to all members of the Program Committee, their colleagues who helped evaluating the submissions, and to the members of the Faculty of Informatics of the University of Debrecen who were involved in the local organization of NCMA 2022.

August 2022

Henning Bordihn
Géza Horváth
György Vaszil

Contents

Paradigms of Reversibility in Reaction Systems

Attila Bagossy
Qogita
London, United Kingdom
attilabagossy@qogita.com

We give an overview of our previous work regarding reaction systems and reversibility by considering different variants of the underlying computational model and the approach to reversibility. We first examine a more conventional form of reversibility, backtracking, implemented in the original model. Then, we introduce a distributed variant called communicating reaction systems, formed by concurrently operating components. As backtracking, a paradigm suited for sequential computation, is overly restrictive for concurrent systems, we conclude the paper by studying the application of causal-consistent reversibility to communicating reaction systems.

1 Introduction

Reactions systems, originating from Ehrenfeucht and Rozenberg [9], form a computational model with the motivation to capture the biochemical processes inside living cells. Computations that correspond to such processes are evolutions of the inner state of the cell, represented by a set of entities. The subsequent evolutions of the inner state, in turn, are guided by the interplay between facilitation and inhibition: the contents of the state set determine whether a reaction is enabled (in other words, facilitated) or not (being inhibited). Computation goes forward by applying enabled reactions to the present state, producing an entirely new set of entities. Such state evolutions take place in the framework of interactive processes, which allow for environmental control over the direction of the computation by injecting entities into each subsequent step. For more details, we refer the reader to [6].

Reversibility allows computation to flow in either direction, so backward execution is just as natural as traditional forward computation. Initially motivated by the work of Landauer [10] and Bennett [5] on heat dissipation and reversible Turing machines, studies in the field of reversibility now extend to many applications, paradigms, and underlying computational models. Inspired by the computational exploration and experimentation that reversibility allows, our previous work considered many different variants of reaction systems and paradigms to make them reversible. This paper gives an overview of this research, from the backtracking reversibility of ordinary reaction systems to the causal-consistent behavior of multicomponent communicating systems introduced by Csuhaj-Varjú and Sethy [7].

2 Preliminaries

In what follows, we recall the essential notions regarding reaction systems. For a more detailed introduction, refer to [9]. Given a set of entities S, a reaction is a triplet $a = (R_a, I_a, P_a)$, where $R_a, I_a, P_a \subseteq S$. R_a is the set of *reactants*, I_a is the set of *inhibitors* while P_a contains the *products* of the reaction. Reactions are required to satisfy $R_a \cap I_a = \emptyset$, $R_a \neq \emptyset$ and $P_a \neq \emptyset$. Then, $\text{rac}(S)$ denotes the set of all reactions over S.

Given two sets of entities, S and $W \subseteq S$, a reaction a *in*rac(S) is *enabled by* W if $R_a \subseteq W$ and $I_a \cap W = \emptyset$. Applying the reaction to W yields its *result on* W defined as $\text{res}_a(W) = P_a$ if a is enabled by W, or $\text{res}_a(W) = \emptyset$ if a is not enabled by W.

We can extend these definitions to sets of reactions as follows. Given a finite set of reactions $A \subseteq \text{rac}(S)$, the *set of all reactions in A enabled by* $W \subseteq S$ is defined as $\text{en}_A(W) = \{a \in A \mid a \text{ is enabled by } W\}$, and the *result of A on W* is defined as $\text{res}_A(W) = \bigcup_{a\in A} \text{res}_a(W)$. For a set of reactions A, we denote by R_A and P_A the union of the reactant sets and product sets, that is $R_A = \cup_{a\in A} R_a$ and $P_A = \cup_{a\in A} P_a$. Furthermore, $\text{EN}_A(S)$ contains sets of reactions in A, where the members of each set are enabled together for a subset of S. Formally $\text{EN}_A(S) = \{E \subseteq A \mid \text{there exists } S' \subseteq S, \text{ such that } \text{en}_A(S') = E\}$. Parallel to the previous definitions, $\text{RES}_A(S)$ contains the result sets of applying every reaction set in $\text{EN}_A(S)$ to the appropriate subsets S, that is, $\text{RES}_A(S) = \{\text{res}_E(S') \mid S' \subseteq S, E \subseteq A, \text{ such that } \text{en}_A(S') = E\}$.

A *reaction system* is a pair $\mathscr{A} = (S,A)$, where S is finite set of entities, called the *background set*, and $A \subseteq \text{rac}(S)$ is the finite *set of reactions*.

An *interactive process* in a reaction system $\mathscr{A} = (S,A)$ is a pair $\pi = (\gamma, \delta)$ of finite sequences, such that $\gamma = C_0, C_1, \ldots C_m$ is the *context sequence* of π, where $C_i \subseteq S$ for al $0 \leq i \leq m$, and $\delta = D_0, D_1, \ldots D_m$ is the *result sequence* of π, where $D_0 = \emptyset$ and $D_i = \text{res}_A(D_{i-1} \cup C_{i-1})$ for all $1 \leq i \leq m$. The union of the sets in γ and δ forms the *state sequence* of π, defined as $\text{sts}(\pi) = W_0^i, W_1^i, \ldots W_m^i$, where $W_i^i = C_i \cup D_i$ for $0 \leq i \leq m$.

In the reaction system $\mathscr{A} = (S,A)$, $W' \subseteq S$, is the *direct predecessor* of $W \subset S$ if $\text{res}_A(W') = W$. The set of direct predecessors of W is denoted by $\text{pred}(W)$. The set W, has a *unique predecessor* if $|\text{pred}(W)| = 1$. Otherwise, W has *multiple predecessors*.

3 Reaction Systems with Backtracking Reversibility

We can regard backtracking as the most traditional form of reversibility: given a sequence of computational steps, we execute an inverse action for each step in reverse order. A simple example is a series of additions and multiplications: we can revert these operations by performing an appropriate subtraction or division in reverse order. Thus, as one can note, the order of computational steps is imperative for their reversal in the case of backtracking. As a result, this paradigm of reversibility relies on two central notions: inverse steps and the concept of the last performed action. For each forward step, one must define its corresponding inverse action (as in our previous example, subtraction is the inverse of addition), and, generally, computations must follow a strict, sequential ordering, so the last action is unambiguous in any given step.

In the case of reaction systems, computations are represented by interactive processes. Each process consists of a finite series of states, where subsequent states result from environmental input and reaction application. Applying the set of enabled reactions produces the so-called result set, which yields the next process state when unioned with the context set from the environment. Consequently, an exact last action is an inherent property of interactive processes, being a linear sequence of states. Thus, implementing backtracking reversibility for reaction systems is mainly a question of inverse steps. While the work of Aman and Ciobanu is concerned with the reverse of single reactions [1], we took a different route by focusing on states and their predecessors.

Bennett's work on reversible Turing machines was motivated by Landauer's argument regarding heat dissipation and information preservation: showing that computing with reversible steps leads to the same results. Hence, Bennett's implementation of backtracking reversibility, Compute-Copy-Uncompute, is primarily concerned with the outcome of the computation. While this paradigm is well-suited for Turing machines, in the case of reaction systems, we wanted our implementation to reflect another characteristic nature of reversibility: allowing exploration and experimentation. Our paradigm of choice, Do-Undo-Redo [13], embraces this goal by allowing one to freely undo any previous computation and proceed by choosing a different computational route.

Undoing, which corresponds to the concept of inverse actions, is based on our notion of unique predecessors: a state has a unique predecessor if we cannot compute it by applying reactions to two or more different predecessor states. Then, inverting the computational step leading to this state is straightforward: a single reaction in which the reactant set is equal to the current state while the product set is equal to the predecessor. Having defined both vital notions of reversibility, last action, and inverse steps, we can now define reversible computation in the form of reversible interactive processes.

Definition 1 ([3], Definition 1)**.** *Let $\mathscr{A}$ be a reaction system and π be an interactive process in $\mathscr{A}$. π is* reversible *if every state in* $\mathrm{sts}(\pi)$ *has a unique predecessor.*

Interactive processes continue with an empty result set when there are no enabled reactions in their present state. By augmenting this empty result set with arbitrary environmental context, we can restart the containing process. As reversal is equivalent to a single inverse reaction producing the predecessor of the current state, undoing such restarting computation is essentially analogous to obtaining "something from nothing". To avoid this scenario, we can define the reversibility of reaction systems by excluding such processes.

Definition 2 ([3], Definition 2)**.** *Let $\mathscr{A}$ be a reaction system and $\pi = (\gamma, \delta)$ be an interactive process in $\mathscr{A}$ such that $\delta = D_0, D_1, \ldots, D_n$. The interactive process π is* non-restarting *if $D_i \neq \emptyset$, $1 \leq i \leq n$. If the opposite holds, then π is* restarting.

Definition 3 ([3], Definition 4)**.** *A reaction system $\mathscr{A}$ is* reversible *if every non-restarting interactive process in $\mathscr{A}$ is reversible.*

Taking the above definition of reversibility, the main result of our work in [3] is formulating the necessary and sufficient conditions for the reversibility of reaction systems while also considering environmental contribution (as opposed to [1]).

Theorem 1 ([3], Theorem 1)**.** *A reaction system $\mathscr{A} = (S, A)$ is* reversible*, if and only if the following conditions hold.*

(1) For all $E_1, E_2 \in EN_A(S)$, $E_1 \neq E_2$ implies $P_{E_1} \neq P_{E_2}$.

(2) For all $W_1, W_2 \subseteq S$, $\mathrm{en}_A(W_1) \neq \emptyset$, $W_1 \neq W_2$ implies $\mathrm{en}_A(W_1) \neq \mathrm{en}_A(W_2)$.

(3) For all $R_1, R_2 \in \mathrm{RES}_A(S)$ such that $R_1 = \mathrm{res}_A(W)$ for some state $W \subseteq S$ which is reachable in $\mathscr{A}$, $R_1 \neq R_2$ implies $R_1 \setminus \Sigma_c \neq R_2 \setminus \Sigma_c$.

The conditions of the above theorem approach the unique predecessor property from three angles: reactions, states, and context. Condition (1) formulates that different sets of enabled reactions must have different products. If two different sets of reactions produced the same entities, then the resulting state consisting of these entities would have multiple predecessors. Condition (2) requires different states (or subsets) of S to enable different reactions. If two states enabled the same set of reactions, then the result set of these reactions would be a state having at least two predecessors. While the former conditions are relatively straightforward, Condition (3), formulating requirements for context sets, is more intricate. It is built on the concept of context alphabet: If we allowed every entity in the background set to be present in the context, then a well-chosen input from the environment would turn a state into one with multiple predecessors. Therefore, we require S to be the union of two, not necessarily disjoint sets, the product alphabet Σ_p (entities that may be produced by reactions), and the context alphabet Σ_c (entities that may appear in context sets). As the initial state of an interactive process consists solely of environmental input, the separation of product and context alphabet gives rise to the notion of *reachable states* in S: those that an interactive process can compute. Then, Condition (3) forbids the case when result sets differ only by entities in the context alphabet. Otherwise, given $R_1 = \mathrm{res}_{E_1}(W_1^i)$ and $R_2 = \mathrm{res}_{E_2}(W_2^i)$ with $R = R_1 \setminus \Sigma_c = R_2 \setminus \Sigma_c$ and $C = (R_1 \cup R_2) \cap \Sigma_c$ we would have that $R \cup C$ has multiple predecessors, namely W_1^i and W_2^i.

4 Communicating Reaction Systems

In this section, we give a concise introduction to communicating reaction systems with direct communication (*cdcR systems*, for short); refer to [7] for more details.

The concept of communicating reaction systems stems from the idea of organizing reaction systems into networks. Such a network is a virtual graph with individual systems as its nodes, called components. Computation is synchronized: each system works according to a global clock. In the case of communicating reaction systems, the graph nature of the construction allows components to exchange entities and reactions in each computational step. A component may freely send its products to any node in the graph by specifying its index in a so-called extended reaction. Thus, while the graph nodes are fixed, the interaction between them is dynamic, dictated by the set of enabled extended reactions.

Remark 1. *In what follows, we will only consider cdcR systems communicating by products (*cdcR(p) systems*) and use the abbreviation of the more general term, cdcR system, to refer also to this variant.*

Given a finite set of entities S, an *extended reaction* is a triplet $a = (R_a, I_a, P_a)$. R_a and I_a are the same as in the case of ordinary systems, while $P_a \subseteq S \times \mathbb{N}^+$ is the set of products with targets. Each *product with target* is a $\rho = (p,t)$ pair, where $p \in S$ is the actual product, while $t \in \mathbb{N}^+$ is the index of the targeted component. The pair (p,t) means that the product p is sent to the component with index t, appearing in its subsequent context set. Then, $\mathrm{erac}(S)$ denotes the set of extended reactions over S.

Given two sets of entities, S and $W \subseteq S$, applying $a \in \mathrm{erac}(S)$ to W yields its *results on* W defined as $\mathrm{res}_a(W) = \{p \mid (p,t) \in P_a\}$ if a is enabled by W, or $\mathrm{res}_a(W) = \emptyset$ if a is not enabled by W.

A *communicating reaction system* (cdcR system, in short) formed by $n \geq 1$ reaction systems (its components) is an $n+1$ tuple $\Delta = (S, \mathscr{A}_1, \ldots, \mathscr{A}_n)$. S is a finite set of entities, called the background set of Δ, and $\mathscr{A}_i = (S, A_i)$ is the ith component of Δ. Each component is a reaction system over the background set S with a finite set of extended reactions $A_i \subseteq \mathrm{erac}(S)$.

Let $\Delta = (S, \mathscr{A}_1, \ldots, \mathscr{A}_n)$ be a cdcR system as above. A *global state* of Δ is an n-tuple $W = (W^1, \ldots W^n)$ where $W^i \subseteq S$, for $1 \leq i \leq n$.

An m-step *interactive communicating process* in a cdcR system is an n-tuple $\Pi = (\pi^1, \ldots, \pi^n)$, where each $\pi^i = ((\gamma^i)_m, (\delta^i)_m)$ is an *interactive process at component* $\mathscr{A}_i$, formed by a pair of finite sequences. $(\gamma^i)_m = C_0^i, C_1^i, \ldots, C_m^i$ is the *context sequence* of π^i, where each set consists of entities received by $\mathscr{A}_i$ from other components, that is, $C_j^i = \bigcup_{1 \leq k \leq n, k \neq i} \{p \mid (p,i) \in \mathrm{res}_{A_k}(W_{j-1}^k)\}$ for $j \geq 1$, and $(\delta^i)_m = D_0^i, D_1^i, \ldots, D_m^i$ is the *result sequence* of π^i, where each set comprises the entities which are not communicated to other components, that is, $D_j^i = \{p \mid (p,i) \in \mathrm{res}_{A_i}(W_{j-1}^i)\}$ for $j \geq 1$. The *state sequence of* π^i is defined as $\mathrm{sts}(\pi^i) = (W^i)_m = W_0^i, W_1^i, \ldots, W_m^i$, where $W_j^i = C_j^i \cup D_j^i$.

5 Approaching Reversibility in Concurrent Models

After fitting ordinary reaction systems with backtracking reversibility and investigating the computational capabilities of this model [3, 2], we wanted to depart from the original definition of reaction systems and see how modifications to the model interact with different approaches to reversibility. cdcR systems, comprised of a network of components, seemed to offer a radically different model, thus providing an ideal host for experimentation.

Nevertheless, while ordinary reaction systems and cdcR systems may have significant differences in shape, they offer the same capabilities from a computational point of view. As shown by Csuhaj-Varjú and Sethy in [7], we can represent any cdcR system in the form of a so-called flattened reaction system, constructed according to the original definition of the model. Hence, without modifications, cdcR systems can be seen as decomposed reaction systems for which backtracking is the most appropriate paradigm of reversibility.

In our introduction to backtracking, we identified its two key concepts: inverse steps and the notion of last performed action. The latter requires a strict ordering of computational steps so that the last step is always unambiguous. This total ordering is a common trait of both ordinary reaction systems and cdcR systems: the computational steps of interactive processes are driven by a single clock. In the case of communicating systems, individual components apply reactions synchronously with each tick of the clock, producing a new global state. Because of the clock-based synchronization of each computational step, the network behaves like a single, sequential system: the only valid backward execution path is the exact reverse ordering of forward steps.

When a computational model lacks the definition of the last action, for example, by introducing concurrent execution, imposing a single valid ordering on backward steps becomes overly restricting. If we can perform forward computations concurrently, we should not place an arbitrary order on reversing them apart from the inherent computational dependencies between appropriate steps. Consequently, independent steps should be reversible in any order. This insight led to the development of new paradigms of reversibility, studying more flexible but still correct orderings of backward steps, such as causal-consistency.

Introduced by Danos and Krivine [8], causal-consistency establishes a link between reversibility, concurrency, and causality. Building on the causal relationships between computational steps, causal-consistency allows us to undo an action once we reverse all of its consequences. Hence, the notions of cause and consequence, underpinning causality, are fundamental to this paradigm. On the contrary, computations with no causal dependencies can be reversed in any order. Therefore, backward execution enjoys the same reordering property as forward execution, resulting in the potential of multiple backward computational traces leading to the same state. Refer to [11] for a comprehensive survey on causal-consistency and its implementations.

6 Communicating Reaction Systems with Causal-Consistent Reversibility

We saw causal-consistency as a paradigm more suited for cdcR systems than backtracking: enabling individual components to compute forward and backward while respecting causal dependencies embraces the networked nature of the model better than strict clock-based synchronization. To adapt causal-consistency to the model, we made two modifications: we removed global synchronization and introduced the notion of blocking. Global synchronization was removed between the components, allowing external control to decide which components shall compute next. Thus, so-called active components apply reactions and consume environmental input, producing new result sets, while idle components retain their contained entities and accept communicated symbols. Blocking, in turn, restricts which components may be active in a given computational step. A component is blocked if it has no enabled reactions. As the absence of enabled reactions would produce the empty result set, we think of blocking as the inability to make further progress. Because producing the empty result set is now impossible, every interactive process, by definition, becomes non-restarting. In what follows, we briefly present our results concerning the reversibility of this model. Refer to [4] for detailed definitions and proofs.

We refer to the above model as distributed cdcR systems or d-cdcR systems for short. Their networked structure is the same as that of cdcR systems, with the difference between the two models being their computational behavior, characterized by the following definition of distributed interactive processes.

Definition 4 ([4], Definition 4)**.** *An m-step* distributed interactive process *is an n-tuple* $\bar{\Pi} = (\bar{\pi}^1, \ldots, \bar{\pi}^n)$ *where each* $\bar{\pi}^i = ((\gamma^i)_m, (\delta^i)_m)$ *is an* interactive process at component $\mathscr{A}_i$ *with* $(\gamma^i)_m = C_0^i, C_1^i, \ldots, C_m^i$ *and* $(\delta^i)_m = D_0^i, D_1^i, \ldots, D_m^i$*, where the following conditions hold. For each component* $\mathscr{A}_i$*, the context sets* C_j^i *(*$0 \le j \le m-1$*) in* $(\gamma^i)_m$ *are as follows:*

$$ComTo_j^i = \bigcup_{\substack{1 \le k \le n,\ k \ne i \\ k \in Active_{j-1}}} \{p \mid (p,i) \in \mathrm{res}_{A_k}(W_{j-1}^k)\} \cup C_j^{env,i},$$

$$C_j^i = \begin{cases} ComTo_j^i, \ \textit{if } i \in Active_{j-1}, \\ ComTo_j^i \cup C_{j-1}^i, \ \textit{otherwise}, \end{cases}$$

where $C_j^{env,i}$ *is the input received by* $\mathscr{A}_i$ *from the environment in the jth step of the interactive process. Furthermore, in each step,* $0 \le j \le m-1$*, there is a set* $Step_j \subseteq \{1, \ldots, n\}$ *such that*

$$Active_j = \{i \mid i \in Step_j \textit{ and } i \notin Block_j,\ 1 \le i \le n\},$$

where $Block_j = \{i \mid \mathrm{res}_{A_i}(W_j^i) = \emptyset\}$.

Then, the result sets D_j^i *(*$0 \le j \le m-1$*) of* $\mathscr{A}_i$ *in* $(\delta^i)_m$ *are as follows:*

$$D_j^i = \begin{cases} \{p \mid (p,i) \in \mathrm{res}_{A_i}(W_{j-1}^i)\}, \ \textit{if } i \in Active_{j-1}, \\ D_{j-1}^i, \ \textit{otherwise}. \end{cases}$$

When defining causal-consistency for d-cdcR systems, we used the framework of Lanese, Philips, and Ulidowski [12], based on labeled transition systems (LTSs). The framework provides a set of axioms that lead to more complex properties once proven for the LTS derived from a given model. Intended to help the development of new causal-consistent models, it eliminates the need for intricate proofs and facilitates the comparison of different models and approaches. Consequently, we translated the individual d-cdcR systems into LTSs.

Definition 5 ([4], Definition 5)**.** *A labeled transition system (LTS) in a triplet* $(Proc, Lab, Tr)$*, where Proc is the set of processes, Lab is the set of labels, and* $Tr \subseteq Proc \times Lab \times Proc$ *is a transition relation. A transition t from P to Q, labelled with a, is denoted as* $t : P \xrightarrow{a} Q$.

Then, the framework imposes the following requirements on the translated LTSs:

- Every transition is reversible.

- There is an irreflexive, symmetric, binary independence relation defined on transitions.

Reverting transitions is directly related to the reversal of reactions and, thus, states. However, determining the predecessor of an individual component's state is insufficient: backward transitions must respect causality by considering the communications between components. Thus, our approach to reversible transitions was twofold. First, we required each component of d-cdcR systems to be locally reversible, ensuring a unique predecessor for each result set. In such a system, each component is reversible according to the requirements of Theorem 1. Second, we recorded the incoming and outgoing communication for every component of the system by introducing the *send* and *recv* stacks into the distributed interactive processes.

Definition 6 ([4], Definition 8)**.** *An m-step* reversible distributed interactive process *is an n-tuple* $\bar{\Pi} = (\bar{\pi}^1, \ldots, \bar{\pi}^n)$ *where each* $\bar{\pi}^i = ((\gamma^i)_m, (\delta^i)_m, (send^i)_m, (recv^i)_m)$ *is an* interactive process at component $\mathscr{A}_i$ *with* $(\gamma^i)_m$ *and* (δ^i) *defined identically as in Definition 4. For each component* $\mathscr{A}_i$*,* $(send^i)_m$ *and* $(recv^i)_m$ *are series of stack-families.*

In a reversible distributed interactive process, we keep track of the communication between the components by putting appropriate entries onto the stacks. As reaction systems are incapable of counting (thanks to their set-based nature), we record the time a communication occurred by placing result sets onto the stacks. Since our definition of reversibility forbids computational cycles, result sets precisely and uniquely determine the timing.

Turning the model of reversible d-cdcR systems into LTSs is now a matter of appropriate processes, transitions, and labels.

Definition 7 ([4], Definition 9)**.** *Let* Δ *be a reversible d-cdcR system, formed by* $n \geq 1$ *components. Then, a process* $P \in Proc$ *in the corresponding LTS is as follows:*

$$P = ((W^1, send^1, recv^1), \ldots, (W^n, send^n, recv^n)),$$

where, for each $1 \leq i \leq n$*,* W^i *is the state of component* $\mathscr{A}_i$ *and* $send^i$ *and* $recv^i$ *denote the* $send^{i,k}$*,* $recv^{k,i}, recv^{env,i}$ *stacks respectively (*$1 \leq k \leq n$*,* $k \neq i$*).*

Definition 8 ([4], Definition 10, 18)**.** *Let* $\Delta = (S, \mathscr{A}^1, \ldots, \mathscr{A}^n)$ *be a reversible d-cdcR system, formed by* $n \geq 1$ *components. Then, a forward label* $a \in Lab$ *in the corresponding LTS is as follows:*

$$a = active; comm,$$

where $active \subseteq \{1, \ldots, n\}$ *contains the indices of active components and* $comm \subseteq \{1, \ldots, n, env\} \times S \times \{1, \ldots, n\}$ *is the set of communicated symbols in the form of* $(sender, symbol, recipient)$ *triplets, such that* $sender \neq receiver$*. A* backward label $\underline{a} \in \underline{Lab}$ *in the corresponding LTS is as follows:*

$$\underline{a} \subseteq \{1, \ldots, n\} \cup \{env_1, \ldots, env_n\},$$

where $i \in \underline{a}$ *corresponds to the reversal of computation in component* $\mathscr{A}_i$*, and* $env_i \in \underline{a}$ *represents the removal of the most recently received environmmental input from component* $\mathscr{A}_i$*.*

We can readily determine the inverse labels for the forward and backward labels defined above. Hence, our LTS satisfies the requirement of reversible transitions. In the form of the framework's second requirement, the independence relation defined on transitions, we establish the connection between causality, concurrency, and reversibility. Our definition of independence revolves around two core concepts: causes and conflicts. As components influence the computational steps of each other by communication, applying reactions to received entities, for example, results in causal dependency: the reactions producing the communicated entities caused the application of reactions in the recipient. We refer to the lack of such causality between subsequent computational steps (and their corresponding transitions) as independence. We can also translate this idea to transitions starting from the same process (conflicts).

Building on the above definition of labels (transitions) and independence, by proving the axioms of [12], we can conclude that d-cdcR systems, with modifications, are causal-consistent reversible.

7 Conclusion

In this paper, we gave a concise overview of our research in the field of reaction systems and reversibility by presenting our implementation of backtracking and causal-consistent reversibility. Future research directions include other paradigms of reversibility, such as out-of-causal-order, as well as mixed-mode networks in which components may perform forward and backward computations in the same step.

References

[1] Bogdan Aman & Gabriel Ciobanu (2018): *Controlled Reversibility in Reaction Systems*. In Marian Gheorghe, Grzegorz Rozenberg, Arto Salomaa & Claudio Zandron, editors: *Membrane Computing*, Springer International Publishing, Cham, pp. 40–53.

[2] Attila Bagossy & György Vaszil (2021): *Transition Graphs of Reversible Reaction Systems*. In Rudolf Freund, Tseren-Onolt Ishdorj, Grzegorz Rozenberg, Arto Salomaa & Claudio Zandron, editors: *Membrane Computing*, Springer International Publishing, Cham, pp. 1–16.

[3] Attila Bagossy & György Vaszil (2020): *Simulating reversible computation with reaction systems*. *Journal of Membrane Computing* 2, pp. 1–15, doi:10.1007/s41965-020-00049-9.

[4] Attila Bagossy & György Vaszil (2022): *Controlled reversibility in communicating reaction systems*. *Theoretical Computer Science* 926, pp. 3–20, doi:10.1016/j.tcs.2022.05.030.

[5] C. H. Bennett (1973): *Logical Reversibility of Computation*. *IBM Journal of Research and Development* 17(6), pp. 525–532.

[6] Robert Brijder, Andrzej Ehrenfeucht, Michael Main & Grzegorz Rozenberg (2011): *A Tour of reaction Systems*. *Int. J. Found. Comput. Sci.* 22, pp. 1499–1517, doi:10.1142/S0129054111008842.

[7] Erzsébet Csuhaj-Varjú & Pramod Kumar Sethy (2021): *Communicating Reaction Systems with Direct Communication*. In Rudolf Freund, Tseren-Onolt Ishdorj, Grzegorz Rozenberg, Arto Salomaa & Claudio Zandron, editors: *Membrane Computing*, Springer International Publishing, Cham, pp. 17–30.

[8] Vincent Danos & Jean Krivine (2004): *Reversible Communicating Systems*. In Philippa Gardner & Nobuko Yoshida, editors: *CONCUR 2004 - Concurrency Theory*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 292–307.

[9] A. Ehrenfeucht & G. Rozenberg (2007): *Reaction Systems*. *Fundam. Inf.* 75(1–4), p. 263–280.

[10] R. Landauer (1961): *Irreversibility and Heat Generation in the Computing Process*. *IBM Journal of Research and Development* 5(3), pp. 183–191.

[11] Ivan Lanese, Claudio Mezzina & Francesco Tiezzi (2014): *Causal-Consistent Reversibility*. *Bull. EATCS* 114, p. 17.

[12] Ivan Lanese, Iain Phillips & Irek Ulidowski (2020): *An Axiomatic Approach to Reversible Computation*. In Jean Goubault-Larrecq & Barbara König, editors: *Foundations of Software Science and Computation Structures*, Springer International Publishing, Cham, pp. 442–461.

[13] Kalyan S. Perumalla (2013): *Introduction to Reversible Computing*. Chapman & Hall/CRC.

Automata for Synchronised Shuffle Expressions

Sabine Broda
sabine.broda@fc.up.pt

António Machiavelo
antonio.machiavelo@fc.up.pt

Nelma Moreira
nelma.moreira@fc.up.pt

Rogério Reis
rogerio.reis@fc.up.pt

CMUP & DM-DCC, Faculdade de Ciências da Universidade do Porto [*]
Porto, Portugal

Several notions of synchronisation in concurrent systems can be modelled by regular shuffle operators. In this paper we consider regular expressions extended with three operators corresponding, to strong, arbitrary, and weak synchronisation. For these expressions, we define a location based position automaton. Furthermore, we show that for expressions with synchronised shuffle operators the partial derivative automaton is still a quotient of the position automaton.

1 Introduction

Several notions of synchronisation in concurrent systems can be modelled by regular shuffle operators. These operations range from the plain shuffle to intersection, which can be seen as two extreme cases, corresponding to pure interleaving and strict synchronisation. If only a subset of letters is allowed to synchronise, several variants of synchronisation can be considered when shuffling two words [2]. In particular, synchronising letters have to always synchronise — strong synchronised shuffle, they can synchronise or not — arbitrary synchronised shuffle, or synchronising letters can optionally be interleaved from one word, but not from the other until the next synchronisation occurs — weak synchronised shuffle. Sulzmann and Thiemann [7] introduced a general synchronised shuffling operator, that subsumes the above three operators, among others. For their general shuffling operator the authors extended the notion of partial derivative and partial derivative automaton ($\mathscr{A}_{\text{PD}}$) [1]. Another conversion from standard regular expressions to automata is the position automaton, of which the partial derivative automaton is a quotient. Positions correspond to the occurrences of alphabet symbols in the expression, and there is a one-to-one correspondence between the set of states in the position automaton and the set of positions. That is no longer true if one considers regular expressions with shufflings. Broda et al. [4] extended the notion of position to shuffle and to intersection by considering tree-like structures, called locations, that keep track of the set of positions, that correspond to the states of the automaton.

In this paper we show that locations can also be used to define a position automaton ($\mathscr{A}_{\text{POS}}$) for the general shuffling operator, more precisely for each of the synchronised shuffle operators. Furthermore, we show that for expressions with synchronised shuffle operators the partial derivative automaton is still a quotient of the position automaton. Most of the proofs omitted here can be found in Broda et al. [5].

[*]This work was partially supported by CMUP, which is financed by national funds through FCT – Fundação para a Ciência e a Tecnologia, I.P., under the project with reference UIDB/00144/2020.

2 Synchronised Shuffle Operators

In this section we review three synchronised shuffle operators, that were studied by Beek et al. [2] and by Sulzmann and Thiemann [7], presenting for each operator two equivalent definitions. Let Σ be a finite set of alphabet symbols and let Σ_w denote the set of alphabet symbols in $w \in \Sigma^\star$.

Strongly Synchronised Shuffle w.r.t. a set Γ Given a set of alphabet symbols $\Gamma \subseteq \Sigma$, the strongly synchronised shuffle of two words w.r.t. Γ imposes synchronisation on all letters of Γ. The *strongly synchronised shuffle* of two words $u, v \in \Sigma^\star$ w.r.t. Γ, and denoted by $u\,{}^{\mathsf{s}}\|_\Gamma\, v$, is the finite set of words defined inductively as follows [7].

$$\varepsilon\,{}^{\mathsf{s}}\|_\Gamma\, v = v\,{}^{\mathsf{s}}\|_\Gamma\, \varepsilon \;=\; \begin{cases} \{v\} & \text{if } \Sigma_v \cap \Gamma = \emptyset, \\ \emptyset & \text{otherwise}, \end{cases}$$

$$\sigma u\,{}^{\mathsf{s}}\|_\Gamma\, \tau v \;=\; \begin{cases} \{\, \sigma w \mid w \in u\,{}^{\mathsf{s}}\|_\Gamma\, v \,\} & \text{if } \sigma = \tau \wedge \sigma \in \Gamma, \\ \emptyset & \text{if } \sigma \neq \tau \wedge \sigma, \tau \in \Gamma, \\ \{\, \sigma w \mid w \in u\,{}^{\mathsf{s}}\|_\Gamma\, \tau v \,\} & \text{if } \sigma \notin \Gamma \wedge \tau \in \Gamma, \\ \{\, \tau w \mid w \in \sigma u\,{}^{\mathsf{s}}\|_\Gamma\, v \,\} & \text{if } \sigma \in \Gamma \wedge \tau \notin \Gamma, \\ \{\sigma w \mid w \in u\,{}^{\mathsf{s}}\|_\Gamma\, \tau v\} \cup \{\, \tau w \mid w \in \sigma u\,{}^{\mathsf{s}}\|_\Gamma\, v\} & \text{if } \sigma, \tau \notin \Gamma. \end{cases}$$

Remark 1. *Note that for $\Gamma = \emptyset$ the operator ${}^{\mathsf{s}}\|_\emptyset$ coincides with the usual shuffle operator $\sqcup\!\sqcup$, given by $u \sqcup\!\sqcup\, \varepsilon = \varepsilon \sqcup\!\sqcup\, u = \{u\}$ and $\sigma u \sqcup\!\sqcup\, \tau v = \{\, \sigma w \mid w \in u \sqcup\!\sqcup\, \tau v\} \cup \{\, \tau w \mid w \in \sigma u \sqcup\!\sqcup\, v\}$.*

Example 1. $abca\,{}^{\mathsf{s}}\|_{\{a\}}\, ada = \{abcda, abdca, adbca\}$.

The following is an equivalent definition of the strongly synchronised shuffle of two words [2].

$$\begin{aligned} u\,{}^{\mathsf{s}}\|_\Gamma\, v = \{x \in (u_1 \sqcup\!\sqcup\, v_1)\sigma_1 \cdots \sigma_{n-1}(u_n \sqcup\!\sqcup\, v_n) \mid n \geq 1 \wedge i \in [n] \wedge \sigma_i \in \Gamma \wedge u_i, v_i \in (\Sigma \setminus \Gamma)^\star \wedge \\ u = u_1\sigma_1 \cdots \sigma_{n-1}u_n \wedge v = v_1\sigma_1 \cdots \sigma_{n-1}v_n \,\}. \end{aligned} \tag{1}$$

Arbitrary Synchronised Shuffle w.r.t. a set Γ Given a set of alphabet symbols $\Gamma \subseteq \Sigma$, the arbitrary synchronised shuffle of two words w.r.t. Γ permits symbols in Γ to synchronise, but does not force their synchronisation. Formally, the *arbitrary synchronised shuffle* of words u and v, denoted by $u\,{}^{\mathsf{a}}\|_\Gamma\, v$, is defined as follows [7].

$$\varepsilon\,{}^{\mathsf{a}}\|_\Gamma\, v = v\,{}^{\mathsf{a}}\|_\Gamma\, \varepsilon \;=\; \{v\},$$

$$\sigma u\,{}^{\mathsf{a}}\|_\Gamma\, \tau v \;=\; \begin{cases} \{\sigma w \mid w \in u\,{}^{\mathsf{a}}\|_\Gamma\, \tau v\} \cup \{\, \tau w \mid w \in \sigma u\,{}^{\mathsf{a}}\|_\Gamma\, v\} & \text{if } \sigma \neq \tau \vee \sigma \notin \Gamma, \\ \{\sigma w \mid w \in u\,{}^{\mathsf{a}}\|_\Gamma\, \tau v\} \cup \{\, \tau w \mid w \in \sigma u\,{}^{\mathsf{a}}\|_\Gamma\, v\} & \\ \quad \cup \{\, \sigma w \mid w \in u\,{}^{\mathsf{a}}\|_\Gamma\, v \,\} & \text{if } \sigma = \tau \wedge \sigma \in \Gamma. \end{cases}$$

Alternatively, we can define ${}^{\mathsf{a}}\|_\Gamma$ as follows [2].

$$\begin{aligned} u\,{}^{\mathsf{a}}\|_\Gamma\, v = \{x \in (u_1 \sqcup\!\sqcup\, v_1)\sigma_1 \cdots \sigma_{n-1}(u_n \sqcup\!\sqcup\, v_n) \mid n \geq 1 \wedge i \in [n] \wedge \sigma_i \in \Gamma \wedge u_i, v_i \in \Sigma^\star \wedge \\ u = u_1\sigma_1 \cdots \sigma_{n-1}u_n \wedge v = v_1\sigma_1 \cdots \sigma_{n-1}v_n \,\}. \end{aligned} \tag{2}$$

Example 2. $ab\,{}^{\mathsf{a}}\|_{\{a\}}\, da = \{abda, adba, adab, dab, daab, daba\}$.

Weak Synchronised Shuffle w.r.t. a set Γ In the *weak synchronised shuffle* of two words u and v, symbols in Γ can be synchronised or optionally be interleaved from one word, but not from the other, until the next synchronisation occurs. Given two words u and v, the definition of $u\,{}^{\mathsf{w}}\|_\Gamma\, v$ resorts to an auxiliary operator $\|_{(\Gamma,\Delta,\Lambda)}$. This operator memorises in Δ (resp. Λ) the elements in Γ, that have been interleaved from u (resp. v) since the last synchronisation has occured. Note that in every step during the computation of $u\,{}^{\mathsf{w}}\|_\Gamma\, v$ the operator $\|_{(\Gamma,\Delta,\Lambda)}$ has the invariant $\Delta\cap\Lambda=\emptyset\wedge\Delta\cup\Lambda\subseteq\Gamma$.

$$\begin{aligned}
u\,{}^{\mathsf{w}}\|_\Gamma\, v &= u\,\|_{(\Gamma,\emptyset,\emptyset)}\, v,\\
\varepsilon\,\|_{(\Gamma,\Delta,\Lambda)}\, v &= \begin{cases}\{v\} & \text{if } \Sigma_v\cap\Delta=\emptyset,\\ \emptyset & \text{otherwise,}\end{cases} \qquad u\,\|_{(\Gamma,\Delta,\Lambda)}\,\varepsilon = \begin{cases}\{u\} & \text{if } \Sigma_u\cap\Lambda=\emptyset,\\ \emptyset & \text{otherwise,}\end{cases}\\
\sigma u\,\|_{(\Gamma,\Delta,\Lambda)}\,\tau v &= \\
&\begin{cases}
\{\,\sigma w \mid w\in u\,\|_{(\Gamma,\Delta,\Lambda)}\,\tau v\,\}\cup\{\,\tau w\mid w\in\sigma u\,\|_{(\Gamma,\Delta,\Lambda)}\, v\,\} & \text{if } \sigma,\tau\notin\Gamma,\\
\{\,\sigma w\mid w\in u\,\|_{(\Gamma,\emptyset,\emptyset)}\, v\,\}\cup\{\,\sigma w\mid w\in u\,\|_{(\Gamma,\Delta\cup\{\sigma\},\Lambda)}\,\tau v\wedge\sigma\notin\Lambda\,\} & \\
\quad\cup\{\,\tau w\mid w\in\sigma u\,\|_{(\Gamma,\Delta,\Lambda\cup\{\tau\})}\, v\wedge\tau\notin\Delta\,\} & \text{if } \sigma=\tau\in\Gamma,\\
\{\,\sigma w\mid w\in u\,\|_{(\Gamma,\Delta\cup\{\sigma\},\Lambda)}\,\tau v\wedge\sigma\notin\Lambda\,\} & \\
\quad\cup\{\,\tau w\mid w\in\sigma u\,\|_{(\Gamma,\Delta,\Lambda\cup\{\tau\})}\, v\wedge\tau\notin\Delta\,\} & \text{if } \sigma\neq\tau,\sigma,\tau\in\Gamma,\\
\{\,\sigma w\mid w\in u\,\|_{(\Gamma,\Delta\cup\{\sigma\},\Lambda)}\,\tau v\wedge\sigma\notin\Lambda\,\} & \\
\quad\cup\{\,\tau w\mid w\in\sigma u\,\|_{(\Gamma,\Delta,\Lambda)}\, v\,\} & \text{if } \sigma\in\Gamma,\tau\notin\Gamma,\\
\{\,\sigma w\mid w\in u\,\|_{(\Gamma,\Delta,\Lambda)}\,\tau v\,\} & \\
\quad\cup\{\,\tau w\mid w\in\sigma u\,\|_{(\Gamma,\Delta,\Lambda\cup\{\tau\})}\, v\wedge\tau\notin\Delta\,\} & \text{if } \sigma\notin\Gamma,\tau\in\Gamma.
\end{cases}
\end{aligned}$$

The operator $\|_{(\Gamma,\Delta,\Lambda)}$ as defined above coincides with the general synchronous shuffling [7] for the weak synchronised shuffle. However, in [7] the authors consider an extension of this operator, where Δ and Λ can be any subsets of Σ. For instance, for $\Delta=\Lambda=\Sigma$ the strong synchronised shuffle operator is obtained. In this paper, we choose to consider the three different forms of shuffling separately, as we feel that in this way functions for the strong and arbitrary synchronised shuffle are easier to understand. We can also have the following alternative definition of $u\,{}^{\mathsf{w}}\|_\Gamma\, v$ from [2].

$$\begin{aligned}u\,{}^{\mathsf{w}}\|_\Gamma\, v=\{\,x\in(u_1\sqcup\!\sqcup v_1)\sigma_1\cdots\sigma_{n-1}(u_n\sqcup\!\sqcup v_n)\mid n\geq 1\wedge i\in[n]\wedge\sigma_i\in\Gamma\wedge u_i,v_i\in\Sigma^\star\wedge\Sigma_{u_i}\cap\Sigma_{v_i}\cap\Gamma=\emptyset\\ \wedge\, u=u_1\sigma_1\cdots\sigma_{n-1}u_n\wedge v=v_1\sigma_1\cdots\sigma_{n-1}v_n\,\}.\end{aligned}\tag{3}$$

Example 3. $abca\,{}^{\mathsf{w}}\|_{\{a,b\}}\, ada=\{abcda,abdca,adbca,abcada,adabca\}$.

Given two languages $L_1,L_2\subseteq\Sigma^\star$ and $\circ\in\{{}^{\mathsf{s}}\|_\Gamma, {}^{\mathsf{a}}\|_\Gamma, {}^{\mathsf{w}}\|_\Gamma\}$ one has, as usual, $L_1\circ L_2=\bigcup_{u\in L_1,v\in L_2}u\circ v$. If L_1 and L_2 are regular, $L_1\circ L_2$ is regular. One can define regular expressions to include the different shuffle operators defined above. The set of extended regular expressions, RE($\|$), contains $\emptyset$ plus all terms generated by the grammar

$$\alpha \;\rightarrow\; \varepsilon\mid\sigma\mid(\alpha+\alpha)\mid(\alpha\cdot\alpha)\mid(\alpha^\star)\mid(\alpha\sqcup\!\sqcup\alpha)\mid(\alpha\,{}^{\mathsf{s}}\|_\Gamma\,\alpha)\mid(\alpha\,{}^{\mathsf{a}}\|_\Gamma\,\alpha)\mid(\alpha\,{}^{\mathsf{w}}\|_\Gamma\,\alpha),$$

where $\sigma\in\Sigma$, $\Gamma\subseteq\Sigma$, ε denotes the empty word and the concatenation operator $\cdot$ is often omitted. Then, the language of an expression $\alpha\in$ RE($\|$) is defined as usual, where for $\circ\in\{\sqcup\!\sqcup, {}^{\mathsf{s}}\|_\Gamma, {}^{\mathsf{a}}\|_\Gamma, {}^{\mathsf{w}}\|_\Gamma\}$ one has $\mathscr{L}(\alpha\circ\beta)=\mathscr{L}(\alpha)\circ\mathscr{L}(\beta)$. We define $\varepsilon(\alpha)$ by $\varepsilon(\alpha)=\mathsf{true}$ if $\varepsilon\in\mathscr{L}(\alpha)$, and $\varepsilon(\alpha)=\mathsf{false}$ otherwise. The *alphabetic size* of α, $|\alpha|_\Sigma$, is the number of occurrences of alphabet symbols in α. Furthermore, we denote the subset of Σ containing the alphabet symbols that occur in α by Σ_α. Considering Remark 1, in the following expressions with the $\sqcup\!\sqcup$ operator will be omitted.

3 A Location Based Position Automaton

In this section we define a position automaton for extended regular expressions with synchronised shuffles, which is based in the notion of location introduced by Broda et al. [4]. Locations are tree-like structures that extend the notion of position to shuffle and intersection.

The Position Automaton. Given a regular expression α with standard operators $(+,\cdot,\star)$, one can mark each occurrence of an alphabet symbol σ with its position in α, reading it from left to right. The resulting regular expression is a *marked* regular expression $\overline{\alpha}$ with all alphabet symbols occurring only once (linear) and belonging to $\Sigma_{\overline{\alpha}}$. Thus, a *position* $i \in [1,|\alpha|_\Sigma]$ corresponds to the symbol σ_i in $\overline{\alpha}$, and thus to exactly one occurrence of σ in α. Given a position i we denote by $\ell(i)$ the symbol $\sigma = \overline{\sigma_i}$. For instance, if $\alpha = a(bb+aba)^\star b$, then $\overline{\alpha} = a_1(b_2b_3+a_4b_5a_6)^\star b_7$ and $\ell(4) = a$. The same notation is used for unmarking, $\overline{\overline{\alpha}} = \alpha$. Let $\mathsf{Pos}(\alpha) = \{1,2,\ldots,|\alpha|_\Sigma\}$, and $\mathsf{Pos}_0(\alpha) = \mathsf{Pos}(\alpha)\cup\{0\}$. Positions were used by Glushkov [6] to define an NFA equivalent to α, usually called the *position* or *Glushkov automaton*, $\mathscr{A}_{\text{POS}}(\alpha)$. Each state of the automaton, except for the initial one, corresponds to a position, and there exists a transition from i to j by σ such that $\overline{\sigma_j} = \sigma$, if σ_i can be followed by σ_j in some word represented by $\overline{\alpha}$. The sets that are used to define the position automaton are $\mathsf{First}(\overline{\alpha}) = \{\, i \mid \exists w \in \Sigma_{\overline{\alpha}}^\star \,.\, \sigma_i w \in \mathscr{L}(\overline{\alpha})\,\}$, $\mathsf{Last}(\overline{\alpha}) = \{\, i \mid \exists w \in \Sigma_{\overline{\alpha}}^\star \,.\, w\sigma_i \in \mathscr{L}(\overline{\alpha})\,\}$ and, given $i \in \mathsf{Pos}(\alpha)$, $\mathsf{Follow}(\overline{\alpha},i) = \{\, j \mid \exists u,v \in \Sigma_{\overline{\alpha}}^\star \,.\, u\sigma_i\sigma_j v \in \mathscr{L}(\overline{\alpha})\,\}$. For the sake of readability, whenever an expression α is not marked, we take $f(\alpha) = f(\overline{\alpha})$, for any function f that has marked expressions as arguments.

Locations. For expressions with shuffle operators there is no more a one-to-one correspondence between the set of states in the $\mathscr{A}_{\text{POS}}$ and the set of positions. More precisely, when we enter a shuffle, we need to know not only one position, but two, since we need to know where we are in the two subwords that are actually shuffled right now. Due to nesting of shuffles, this means that we have to store a tree of positions – the locations. Here, we define locations for expressions in $\mathfrak{R}$. For operators ${}^{\mathsf{s}}\|_\Gamma$ and ${}^{\mathsf{a}}\|_\Gamma$ locations are defined as for $\sqcup\!\sqcup$. The definition of locations for expressions of the form $\alpha\, {}^{\mathsf{w}}\|_\Gamma\, \beta$ will have additional parameters corresponding to the sets Δ and Λ above. Additionally, we annotate each location p with a Boolean b, whose value is true iff the location belongs to Last. The set of *annotated locations* $\mathsf{Loc}(\alpha) = \mathsf{Loc}(\overline{\alpha})$ is inductively defined on the structure of the expression $\overline{\alpha}$ as follows

$$\begin{aligned}
\mathsf{Loc}(\varepsilon) &= \emptyset,\ \mathsf{Loc}(\sigma_i) = \{i : \mathsf{true}\},\ \mathsf{Loc}(\alpha^\star) = \mathsf{Loc}(\alpha),\\
\mathsf{Loc}(\alpha_1+\alpha_2) &= \mathsf{Loc}(\alpha_1)\cup\mathsf{Loc}(\alpha_2),\\
\mathsf{Loc}(\alpha_1\alpha_2) &= \begin{cases}\mathsf{Loc}(\alpha_1)\cup\mathsf{Loc}(\alpha_2), & \varepsilon(\alpha_2)=\mathsf{true},\\ \{\, p:\mathsf{false} \mid p:\mathsf{b}\in\mathsf{Loc}(\alpha_1)\,\}\cup\mathsf{Loc}(\alpha_2), & \text{otherwise,}\end{cases}\\
\mathsf{Loc}(\alpha_1 \sqcup\!\sqcup \alpha_2) &= \mathsf{Loc}(\alpha_1\, {}^{\mathsf{s}}\|_\Gamma\, \alpha_2) = \mathsf{Loc}(\alpha_1\, {}^{\mathsf{a}}\|_\Gamma\, \alpha_2)\\
&= \{\,(p,0):\mathsf{b}\wedge\varepsilon(\alpha_2) \mid p:\mathsf{b}\in\mathsf{Loc}(\alpha_1)\,\}\\
&\cup \{\,(0,q):\mathsf{b}\wedge\varepsilon(\alpha_1) \mid q:\mathsf{b}\in\mathsf{Loc}(\alpha_2)\,\}\\
&\cup \{\,(p,q):\mathsf{b}_1\wedge\mathsf{b}_2 \mid p:\mathsf{b}_1\in\mathsf{Loc}(\alpha_1)\wedge q:\mathsf{b}_2\in\mathsf{Loc}(\alpha_2)\,\},\\
\mathsf{Loc}(\alpha_1\, {}^{\mathsf{w}}\|_\Gamma\, \alpha_2) &= \{\,(p^\Delta,0^\emptyset):\mathsf{b}\wedge\varepsilon(\alpha_2) \mid p:\mathsf{b}\in\mathsf{Loc}(\alpha_1)\wedge\Delta\subseteq\Gamma\,\}\\
&\cup \{\,(0^\emptyset,q^\Lambda):\mathsf{b}\wedge\varepsilon(\alpha_1) \mid q:\mathsf{b}\in\mathsf{Loc}(\alpha_2)\wedge\Lambda\subseteq\Gamma\,\}\\
&\cup \{\,(p^\Delta,q^\Lambda):\mathsf{b}_1\wedge\mathsf{b}_2 \mid p:\mathsf{b}_1\in\mathsf{Loc}(\alpha_1)\wedge q:\mathsf{b}_2\in\mathsf{Loc}(\alpha_2),\\
&\wedge\ \Delta,\Lambda\subseteq\Gamma\ \wedge\ \Delta\cap\Lambda=\emptyset\,\}.
\end{aligned}$$

It follows from this definition, that $p:\mathsf{b}, p:\mathsf{b}' \in \mathsf{Loc}(\alpha)$ implies $\mathsf{b}=\mathsf{b}'$. We will omit the Boolean of an annotated location whenever convenient. Note that each location p in α is either a position $i \in \mathsf{Pos}(\alpha)$, or of the form $(0,q)$, $(p,0)$, (p,q), $(0^{\emptyset}, q^{\Lambda})$, $(p^{\Delta}, 0^{\emptyset})$, or $(p^{\Delta}, q^{\Lambda})$, where p,q are also locations in α and $\Delta, \Lambda \subseteq \Sigma$. *The set of positions of a location* p, $\mathsf{lp}(p)$, is defined inductively by

$$\begin{aligned}\mathsf{lp}(i) &= \{i\},\\ \mathsf{lp}((0^{\emptyset}, p^{\Lambda})) = \mathsf{lp}((0,p)) = \mathsf{lp}((p^{\Delta}, 0^{\emptyset})) = \mathsf{lp}((p,0)) &= \mathsf{lp}(p),\\ \mathsf{lp}((p^{\Delta}, q^{\Lambda})) = \mathsf{lp}((p,q)) &= \mathsf{lp}(p) \cup \mathsf{lp}(q).\end{aligned}$$

Location Automaton for RE($\|$) Given $\alpha \in \mathrm{RE}(\|)$, the states in the position automaton will be labelled by the elements in $\mathsf{Loc}(\alpha)$, except for the initial state labelled by 0. The set $\mathsf{First}(\alpha) \subseteq \Sigma \times \mathsf{Loc}(\alpha)$ is defined as follows

$$\begin{aligned}
\mathsf{First}(\varepsilon) = \emptyset, \quad \mathsf{First}(\sigma_i) &= \{(\overline{\sigma_i}, i)\}, \quad \mathsf{First}(\alpha^{\star}) = \mathsf{First}(\alpha),\\
\mathsf{First}(\alpha_1 + \alpha_2) &= \mathsf{First}(\alpha_1) \cup \mathsf{First}(\alpha_2),\\
\mathsf{First}(\alpha_1\alpha_2) &= \begin{cases}\mathsf{First}(\alpha_1) \cup \mathsf{First}(\alpha_2), & \text{if } \varepsilon(\alpha_1) = \mathsf{true},\\ \mathsf{First}(\alpha_1), & \text{otherwise.}\end{cases}\\
\mathsf{First}(\alpha_1 \,{}^{\mathsf{s}}\|_{\Gamma}\, \alpha_2) &= \{\, (\sigma,(p,0)) \mid \sigma \notin \Gamma \wedge (\sigma,p) \in \mathsf{First}(\alpha_1)\,\} \cup \{\, (\sigma,(0,q)) \mid \sigma \notin \Gamma \wedge (\sigma,q) \in \mathsf{First}(\alpha_2)\,\}\\
&\cup \{\, (\sigma,(p,q)) \mid \sigma \in \Gamma \wedge (\sigma,p) \in \mathsf{First}(\alpha_1) \wedge (\sigma,q) \in \mathsf{First}(\alpha_2)\,\},\\
\mathsf{First}(\alpha_1 \,{}^{\mathsf{a}}\|_{\Gamma}\, \alpha_2) &= \{\, (\sigma,(p,0)) \mid (\sigma,p) \in \mathsf{First}(\alpha_1)\,\} \cup \{\, (\sigma,(0,q)) \mid (\sigma,q) \in \mathsf{First}(\alpha_2)\,\}\\
&\cup \{\, (\sigma,(p,q)) \mid \sigma \in \Gamma \wedge (\sigma,p) \in \mathsf{First}(\alpha_1), (\sigma,q) \in \mathsf{First}(\alpha_2)\,\},\\
\mathsf{First}(\alpha_1 \,{}^{\mathsf{w}}\|_{\Gamma}\, \alpha_2) &= \{\, (\sigma,(p^{\emptyset},0^{\emptyset})) \mid \sigma \notin \Gamma \wedge (\sigma,p) \in \mathsf{First}(\alpha_1)\,\} \cup \{\, (\sigma,(0^{\emptyset},q^{\emptyset})) \mid \sigma \notin \Gamma \wedge (\sigma,q) \in \mathsf{First}(\alpha_2)\,\}\\
&\cup \{\, (\sigma,(p^{\emptyset},q^{\emptyset})) \mid \sigma \in \Gamma \wedge (\sigma,p) \in \mathsf{First}(\alpha_1) \wedge (\sigma,q) \in \mathsf{First}(\alpha_2)\,\}\\
&\cup \{\, (\sigma,(p^{\{\sigma\}},0^{\emptyset})) \mid \sigma \in \Gamma \wedge (\sigma,p) \in \mathsf{First}(\alpha_1)\,\}\\
&\cup \{\, (\sigma,(0^{\emptyset},q^{\{\sigma\}})) \mid \sigma \in \Gamma \wedge (\sigma,q) \in \mathsf{First}(\alpha_2)\,\}.
\end{aligned}$$

Note that the definition of $\mathsf{First}(\alpha_1 \sqcup\!\sqcup\, \alpha_2)$ given in [4] coincides precisely with $\mathsf{First}(\alpha_1 \,{}^{\mathsf{s}}\|_{\emptyset}\, \alpha_2)$.

Example 4. *For* $\alpha = ((ab)^{\star} \,{}^{\mathsf{s}}\|_{\{a\}}\, (ca)^{\star}) \,{}^{\mathsf{w}}\|_{\{b\}}\, (bc)^{\star}$ *one has*

$$\begin{aligned}
&\mathsf{First}((ab)^{\star}) = \{(a,1)\},\ \mathsf{First}((ca)^{\star}) = \{(c,3)\},\ \mathsf{First}((ab)^{\star} \,{}^{\mathsf{s}}\|_{\{a\}}\, (ca)^{\star}) = \{(c,(0,3))\},\\
&\mathsf{First}((bc)^{\star}) = \{(b,5)\}, \mathsf{First}(((ab)^{\star} \,{}^{\mathsf{s}}\|_{\{a\}}\, (ca)^{\star}) \,{}^{\mathsf{w}}\|_{\{b\}}\, (bc)^{\star}) = \{(c,((0,3)^{\emptyset},0^{\emptyset})),(0^{\emptyset},5^{b})\}.
\end{aligned}$$

Lemma 1. *Given* $\alpha \in \mathrm{RE}(\|)$*, if there is some* $\sigma w \in \mathscr{L}(\alpha)$*, then there is a location* p *such that* $(\sigma,p) \in \mathsf{First}(\alpha)$.

The set $\mathsf{Last}(\alpha) \subseteq \mathsf{Loc}(\alpha)$ is defined by $\mathsf{Last}(\alpha) = \{\, p \mid p : \mathsf{true} \in \mathsf{Loc}(\alpha)\,\}$. Furthermore, let $\mathsf{Last}_0(\alpha) = \mathsf{Last}(\alpha) \cup \{0\}$ if $\varepsilon(\alpha) = \mathsf{true}$, and $\mathsf{Last}_0(\alpha) = \mathsf{Last}(\alpha)$ otherwise.

Finally, we define $\mathsf{Follow} : \mathrm{RE}(\|) \times \mathsf{Loc}_0(\alpha) \to 2^{\Sigma_{\alpha} \times \mathsf{Loc}(\alpha)}$ by setting $\mathsf{Follow}(\alpha, 0) = \mathsf{First}(\alpha)$, and for $p, p_1, q_1 \in \mathsf{Loc}_0(\alpha)$,

$$\mathsf{Follow}(\alpha_1 + \alpha_2, p) = \begin{cases} \mathsf{Follow}(\alpha_1, p), & \text{if } p \in \mathsf{Loc}(\alpha_1), \\ \mathsf{Follow}(\alpha_2, p), & \text{if } p \in \mathsf{Loc}(\alpha_2), \end{cases}$$

$$\mathsf{Follow}(\alpha_1\alpha_2, i) = \begin{cases} \mathsf{Follow}(\alpha_1, p), & \text{if } p \in \mathsf{Loc}(\alpha_1) \setminus \mathsf{Last}(\alpha_1), \\ \mathsf{Follow}(\alpha_1, p) \cup \mathsf{First}(\alpha_2), & \text{if } p \in \mathsf{Last}(\alpha_1), \\ \mathsf{Follow}(\alpha_2, p), & \text{if } p \in \mathsf{Loc}(\alpha_2), \end{cases}$$

$$\mathsf{Follow}(\alpha^\star, p) = \begin{cases} \mathsf{Follow}(\alpha, p), & \text{if } p \notin \mathsf{Last}(\alpha), \\ \mathsf{Follow}(\alpha, p) \cup \mathsf{First}(\alpha), & \text{otherwise}, \end{cases}$$

$$\begin{aligned}
\mathsf{Follow}(\alpha_1 {}^{\mathsf{s}}\|_\Gamma \alpha_2, (p_1, q_1)) = {} & \{ (\sigma, (p_2, q_1)) \mid (\sigma, p_2) \in \mathsf{Follow}(\alpha_1, p_1) \wedge \sigma \notin \Gamma \} \\
& \cup \{ (\sigma, (p_1, q_2)) \mid (\sigma, q_2) \in \mathsf{Follow}(\alpha_2, q_1) \wedge \sigma \notin \Gamma \} \\
& \cup \{ (\sigma, (p_2, q_2)) \mid (\sigma, p_2) \in \mathsf{Follow}(\alpha_1, p_1) \wedge (\sigma, q_2) \in \mathsf{Follow}(\alpha_2, q_1) \wedge \sigma \in \Gamma \}, \\
\mathsf{Follow}(\alpha_1 {}^{\mathsf{a}}\|_\Gamma \alpha_2, (p_1, q_1)) = {} & \{ (\sigma, (p_2, q_1)) \mid (\sigma, p_2) \in \mathsf{Follow}(\alpha_1, p_1) \} \\
& \cup \{ (\sigma, (p_1, q_2)) \mid (\sigma, q_2) \in \mathsf{Follow}(\alpha_2, q_1) \} \\
& \cup \{ (\sigma, (p_2, q_2)) \mid (\sigma, p_2) \in \mathsf{Follow}(\alpha_1, p_1) \wedge (\sigma, q_2) \in \mathsf{Follow}(\alpha_2, q_1) \wedge \sigma \in \Gamma \}, \\
\mathsf{Follow}(\alpha_1 {}^{\mathsf{w}}\|_\Gamma \alpha_2, (p_1^\Delta, q_1^\Lambda)) = {} & \{ (\sigma, (p_2^\Delta, q_1^\Lambda)) \mid (\sigma, p_2) \in \mathsf{Follow}(\alpha_1, p_1) \wedge \sigma \notin \Gamma \} \\
& \cup \{ (\sigma, (p_1^\Delta, q_2^\Lambda)) \mid (\sigma, q_2) \in \mathsf{Follow}(\alpha_2, q_1) \wedge \sigma \notin \Gamma \} \\
& \cup \{ (\sigma, (p_2^\emptyset, q_2^\emptyset)) \mid (\sigma, p_2) \in \mathsf{Follow}(\alpha_1, p_1) \wedge (\sigma, q_2) \in \mathsf{Follow}(\alpha_1, q_1) \wedge \sigma \in \Gamma \} \\
& \cup \{ (\sigma, (p_2^{\Delta,\sigma}, q_1^\Lambda)) \mid (\sigma, p_2) \in \mathsf{Follow}(\alpha_1, p_1) \wedge \sigma \in \Gamma \wedge \sigma \notin \Lambda \} \\
& \cup \{ (\sigma, (p_1^\Delta, q_2^{\Lambda,\sigma})) \mid (\sigma, q_2) \in \mathsf{Follow}(\alpha_2, q_1) \wedge \sigma \in \Gamma \wedge \sigma \notin \Delta \}.
\end{aligned}$$

For a set $S \subseteq \Sigma \times \mathsf{Loc}(\alpha)$ and $\sigma \in \Sigma$, let $\mathsf{Select}(S, \sigma) = \{ p \mid (\sigma, p) \in S \}$. The *position automaton* for $\alpha \in \mathrm{RE}(\|)$ is

$$\mathscr{A}_{\mathrm{POS}}(\alpha) = \langle \mathsf{Loc}_0(\alpha), \Sigma, \delta_{\mathrm{POS}}, 0, \mathsf{Last}_0(\alpha) \rangle,$$

where $\delta_{\mathrm{POS}}(p, \sigma) = \mathsf{Select}(\mathsf{Follow}(\alpha, p), \sigma)$, for $p \in \mathsf{Loc}_0(\alpha), \sigma \in \Sigma$. The correctness of this construction follows from the following two lemmas.

Lemma 2. *Given $\alpha \in \mathrm{RE}(\|)$, if $x = \sigma_1 \cdots \sigma_n \in \mathscr{L}(\alpha)$ ($n \geq 1$), then there is a sequence $p_0, p_1, \ldots, p_n \in \mathsf{Loc}_0(\alpha)$, such that $p_0 = 0$ and $(\sigma_i, p_i) \in \mathsf{Follow}(\alpha, p_{i-1})$, for $1 \leq i \leq n$ and $p_n \in \mathsf{Last}(\alpha)$.*

Lemma 3. *Given $\alpha \in \mathrm{RE}(\|)$, if there are $r_0 = 0, r_1, \ldots, r_n \in \mathsf{Loc}_0(\alpha)$ where $n \geq 1$, and $\sigma_1, \ldots, \sigma_n \in \Sigma$ such that $(\sigma_i, r_i) \in \mathsf{Follow}(\alpha, r_{i-1})$, $1 \leq i \leq n$ and $r_n \in \mathsf{Last}(\alpha)$, then $\sigma_1 \cdots \sigma_n \in \mathscr{L}(\alpha)$.*

Proposition 4. $\mathscr{L}(\mathscr{A}_{\mathrm{POS}}(\alpha)) = \mathscr{L}(\alpha)$.

4 $\mathscr{A}_{\mathrm{PD}}$ as a quotient of $\mathscr{A}_{\mathrm{POS}}$

In this section, we relate the partial derivative automaton defined by Sulzmann and Thiemann [7] and the position automaton defined in Section 3. For standard regular expressions extended with the shuffle operator $\sqcup\!\sqcup$, the former has been shown to be a quotient of the latter [4]. Here, we extend this result for the three synchronised shuffles. Sulzmann and Thiemann [7] defined the set of partial derivatives of an expression α w.r.t. an alphabet symbol $\sigma \in \Sigma$, denoted by $\partial_\sigma(\alpha)$, for their general shuffle operator. The set of partial derivatives w.r.t. a word $w\sigma$ is then defined as usual, by $\partial_{w\sigma}(\alpha) = \partial_\sigma(\partial_w(\alpha))$. However, since the parameters of the general operator change in each step of derivation, it is not straightforward to express the set of partial derivatives w.r.t. a word w in terms of the partial derivatives w.r.t. subwords of w. In Lemma 5 we obtain such a result for the operators ${}^{\mathsf{s}}\|_\Gamma$, ${}^{\mathsf{a}}\|_\Gamma$ and $\|_{(\Gamma,\Delta,\Lambda)}$. This relation is crucial to show that $\mathscr{A}_{\mathrm{PD}}(\alpha)$ is a quotient of $\mathscr{A}_{\mathrm{POS}}(\alpha)$. The proof of this result follows the one in [4]. To each location $p \in \mathsf{Loc}(\alpha)$ we associate a unique partial derivative of α. This expression is denoted by $\mathsf{c}(\alpha,p)$ and called the c-*continuation of p in α*. Then, the partial derivative automaton $\mathscr{A}_{\mathrm{PD}}(\alpha)$ is obtained by merging in $\mathscr{A}_{\mathrm{POS}}(\alpha)$ states (locations) with the same c-continuation.

In the following, we consider regular expressions with the auxiliary operator $\|_{(\Gamma,\Delta,\Lambda)}$, where we take ${}^{\mathsf{w}}\|_\Gamma$ as an abbreviation of $\|_{(\Gamma,\emptyset,\emptyset)}$. Moreover we also consider RE($\|$) extended with expressions with those operators. The derivative of an expression $\alpha \in \mathrm{RE}(\|)$ by a symbol σ is defined as follows

$$\begin{aligned}
\partial_\sigma(\emptyset) = \partial_\sigma(\varepsilon) = \emptyset, \quad \partial_\sigma(\alpha^\star) = \partial_\sigma(\alpha)\alpha^\star, \quad \partial_\sigma(\sigma') = (\sigma = \sigma')\{\varepsilon\}\\
\partial_\sigma(\alpha+\beta) = \partial_\sigma(\alpha) \cup \partial_\sigma(\beta), \quad \partial_\sigma(\alpha\beta) = \partial_\sigma(\alpha)\beta \cup \varepsilon(\alpha)\partial_\sigma(\beta)\\
\partial_\sigma(\alpha\,{}^{\mathsf{s}}\|_\Gamma\,\beta) &= \begin{cases} \partial_\sigma(\alpha)\,{}^{\mathsf{s}}\|_\Gamma\,\partial_\sigma(\beta), & \text{if } \sigma \in \Gamma,\\ \partial_\sigma(\alpha)\,{}^{\mathsf{s}}\|_\Gamma\,\{\beta\} \cup \{\alpha\}\,{}^{\mathsf{s}}\|_\Gamma\,\partial_\sigma(\beta), & \text{otherwise,} \end{cases}\\
\partial_\sigma(\alpha\,{}^{\mathsf{a}}\|_\Gamma\,\beta) &= (\sigma \in \Gamma)\partial_\sigma(\alpha)\,{}^{\mathsf{a}}\|_\Gamma\,\partial_\sigma(\beta) \;\cup\; \partial_\sigma(\alpha)\,{}^{\mathsf{a}}\|_\Gamma\,\{\beta\} \;\cup\; \{\alpha\}\,{}^{\mathsf{a}}\|_\Gamma\,\partial_\sigma(\beta),\\
\partial_\sigma(\alpha\,\|_{(\Gamma,\Delta,\Lambda)}\,\beta) &= \begin{cases} \partial_\sigma(\alpha)\,\|_{(\Gamma,\Delta,\Lambda)}\{\beta\} \cup \{\alpha\}\,\|_{(\Gamma,\Delta,\Lambda)}\partial_\sigma(\beta), & \text{if } \sigma \notin \Gamma,\\ \partial_\sigma(\alpha)\,\|_{(\Gamma,\emptyset,\emptyset)}\partial_\sigma(\beta) \;\cup\; (\sigma \notin \Lambda)\partial_\sigma(\alpha)\,\|_{(\Gamma,\Delta\cup\{\sigma\},\Lambda)}\{\beta\} \cup & \\ \qquad \cup (\sigma \notin \Delta)\{\alpha\}\,\|_{(\Gamma,\Delta,\Lambda\cup\{\sigma\})}\partial_\sigma(\beta), & \text{otherwise,} \end{cases}
\end{aligned}$$

where, for any $S,T \subseteq \mathrm{RE}(\|)\setminus\{\emptyset\}$ we define $S \circ T = \{\,\alpha \circ \beta \mid \alpha \in S \wedge \beta \in T\,\}$, for $\circ \in \{\,{}^{\mathsf{s}}\|_\Gamma, {}^{\mathsf{a}}\|_\Gamma, \|_{(\Gamma,\Delta,\Lambda)}\,\}$, $S\emptyset = \emptyset S = \emptyset$, $S\varepsilon = \{\varepsilon\}S = S$, and for $\alpha' \neq \varepsilon, \emptyset$, we have $S\alpha' = \{\,\alpha\alpha' \mid \alpha \in S \wedge \alpha \neq \varepsilon\,\} \cup \{\,\alpha' \mid \exists \varepsilon \in S\}$. Moreover, $\mathsf{b}S = S$ if condition b is true, and $\mathsf{b}S = \emptyset$ otherwise.

As usual, the set of partial derivatives of $\alpha \in \mathrm{RE}(\|)$ w.r.t. a word $w \in \Sigma^\star$ is inductively defined by $\partial_\varepsilon(\alpha) = \{\alpha\}$ and $\partial_{w\sigma}(\alpha) = \partial_\sigma(\partial_w(\alpha))$, where, given a set $S \subseteq \mathrm{RE}(\|)$, $\partial_\sigma(S) = \bigcup_{\alpha\in S}\partial_\sigma(\alpha)$. Moreover, $\mathscr{L}(\partial_w(\alpha)) = \{\,w_1 \mid ww_1 \in \mathscr{L}(\alpha)\,\}$. Let $\partial(\alpha) = \bigcup_{w\in\Sigma^\star}\partial_w(\alpha)$, and $\partial^+(\alpha) = \bigcup_{w\in\Sigma^+}\partial_w(\alpha)$. The partial derivative automaton of $\alpha \in \mathrm{RE}(\|)$ is

$$\mathscr{A}_{\mathrm{PD}}(\alpha) = \langle \partial(\alpha), \Sigma, \{\alpha\}, \delta_{\mathrm{PD}}, F_{\mathrm{PD}}\rangle,$$

with $F_{\mathrm{PD}} = \{\,\beta \in \partial(\alpha) \mid \varepsilon(\beta) = \mathsf{true}\,\}$ and $\delta_{\mathrm{PD}}(\beta,\sigma) = \partial_\sigma(\beta)$, for $\beta \in \partial(\alpha)$, $\sigma \in \Sigma$.

Lemma 5. *For $\alpha,\beta \in \mathrm{RE}(\|)$, $w \in \Sigma^+$, and $\circ \in \{\,{}^{\mathsf{s}}\|_\Gamma, {}^{\mathsf{a}}\|_\Gamma\,\}$ the following holds*

$$\begin{aligned}
\partial_w(\alpha \circ \beta) &= \bigcup_{w \in u \circ v} (\partial_u(\alpha) \circ \partial_v(\beta)),\\
\partial_w(\alpha\,\|_{(\Gamma,\Delta,\Lambda)}\,\beta) &= \bigcup_{\substack{w \in (u_1 \sqcup\!\sqcup\, v_1)\sigma_1\cdots\sigma_{n-1}(u_n \sqcup\!\sqcup\, v_n)\\ \Sigma_{u_i}\cap\Sigma_{v_i}\cap\Gamma=\emptyset,\Sigma_{u_1}\cap\Lambda=\emptyset,\Sigma_{v_1}\cap\Delta=\emptyset}} (\partial_u(\alpha)\,\|_{(\Gamma,\Gamma_{u_n},\Gamma_{v_n})}\partial_v(\beta)),
\end{aligned}$$

where $u,v \in \Sigma^\star$, $u = u_1\sigma_1\cdots\sigma_{n-1}u_n$, $v = v_1\sigma_1\cdots\sigma_{n-1}v_n$ and $\sigma_i \in \Gamma$.

Broda et al. [4] showed that for regular expressions with the shuffle operator $\sqcup\!\sqcup$, given a location $p \in \mathsf{Loc}(\alpha)$, there exists a unique expression $\mathsf{c}(\overline{\alpha},p)$, called the c-continuation of p in $\overline{\alpha}$, such that for all $w\sigma_i \in \Sigma^+_{\overline{\alpha}}$ either $\partial_{w\sigma_i}(\overline{\alpha}) = \emptyset$, or there exists a location $p \in \mathsf{Loc}(\alpha)$, with $i \in \mathsf{lp}(p)$, such that $\partial_{w\sigma_i}(\overline{\alpha}) = \{\mathsf{c}(\alpha,p)\}$. Let $\mathsf{c}(\overline{\alpha},0) = \alpha$. For $\alpha \in \mathrm{RE}(\|)$, the c-*continuation* $\mathsf{c}(\overline{\alpha},p)$ of a location $p \in \mathsf{Loc}(\alpha)$ in $\overline{\alpha}$ is defined by the following.

$$
\begin{aligned}
&\mathsf{c}(\sigma_i, i) = \varepsilon, \mathsf{c}(\alpha^\star, p) = \mathsf{c}(\alpha, p)\alpha^\star, \\
&\mathsf{c}(\alpha_1 + \alpha_2, p) = \begin{cases} \mathsf{c}(\alpha_1, p), & \text{if } p \in \mathsf{Loc}(\alpha_1), \\ \mathsf{c}(\alpha_2, p), & \text{if } p \in \mathsf{Loc}(\alpha_2), \end{cases} \quad \begin{aligned} &\mathsf{c}(\alpha_1 {}^{\mathsf{s}}\|_\Gamma \alpha_2, (p_1, p_2)) = \mathsf{c}(\alpha_1, p_1) {}^{\mathsf{s}}\|_\Gamma \mathsf{c}(\alpha_2, p_2), \\ &\mathsf{c}(\alpha_1 {}^{\mathsf{a}}\|_\Gamma \alpha_2, (p_1, p_2)) = \mathsf{c}(\alpha_1, p_1) {}^{\mathsf{a}}\|_\Gamma \mathsf{c}(\alpha_2, p_2), \end{aligned} \\
&\mathsf{c}(\alpha_1 \alpha_2, p) = \begin{cases} \mathsf{c}(\alpha_1, p)\alpha_2, & \text{if } p \in \mathsf{Loc}(\alpha_1), \\ \mathsf{c}(\alpha_2, p), & \text{if } p \in \mathsf{Loc}(\alpha_2), \end{cases} \quad \mathsf{c}(\alpha_1 {}^{\mathsf{w}}\|_\Gamma \alpha_2, (p_1^\Delta, p_2^\Lambda)) = \mathsf{c}(\alpha_1, p_1) \|_{(\Gamma,\Delta,\Lambda)} \mathsf{c}(\alpha_2, p_2).
\end{aligned}
$$

We can now relate the set of c-continuations with $\partial^+(\overline{\alpha})$, characterise the ones that correspond to final states, and relate derivatives of $\mathsf{c}(\overline{\alpha}, p)$ with $\mathsf{Follow}(\alpha, p)$.

Proposition 6. *For* $\alpha \in \mathrm{RE}(\|)$*,*

1. $\partial^+(\overline{\alpha}) \subseteq \{\, \mathsf{c}(\overline{\alpha}, p) \mid p \in \mathsf{Loc}(\alpha) \,\}$*,*
2. $\varepsilon(\mathsf{c}(\overline{\alpha}, p)) = \varepsilon \iff p \in \mathsf{Last}(\alpha)$*, for* $p \in \mathsf{Loc}(\alpha)$*,*
3. *for* $p \in \mathsf{Loc}_0(\alpha)$ *and* $\sigma_i \in \Sigma_{\overline{\alpha}}$*, one has*

$$\beta \in \partial_{\sigma_i}(\mathsf{c}(\overline{\alpha}, p)) \iff \exists q \in \mathsf{Loc}(\alpha) : \beta = \mathsf{c}(\overline{\alpha}, q) \wedge (\overline{\sigma_i}, q) \in \mathsf{Follow}(\alpha, p).$$

Then, we can construct a quotient automaton of $\mathscr{A}_{\mathrm{POS}}(\alpha)$, isomorphic to $\mathscr{A}_{\mathrm{PD}}(\alpha)$, exactly as done in [4]. In fact, the proof presented there, stating the correctness of the construction, relies solely on Proposition 6. Consider the equivalence relation $\equiv_o \subseteq \mathsf{Loc}_0(\alpha) \times \mathsf{Loc}_0(\alpha)$ defined by $p \equiv_o q$ iff $\mathsf{c}(\overline{\alpha}, p) = \mathsf{c}(\overline{\alpha}, q)$. This relation is right-invariant w.r.t. $\mathscr{A}_{\mathrm{POS}}(\alpha)$ and $\mathscr{A}_{\mathrm{POS}}(\alpha)/{\equiv_o} \simeq \overline{\mathscr{A}_{\mathrm{PD}}(\overline{\alpha})}$. Here, $\overline{\mathscr{A}_{\mathrm{PD}}(\overline{\alpha})}$ denotes the automaton obtained from $\mathscr{A}_{\mathrm{PD}}(\overline{\alpha})$ by unmarking the labels of transitions. The isomorphism φ_c between states of $\mathscr{A}_{\mathrm{POS}}(\alpha)/{\equiv_o}$ and states of $\overline{\mathscr{A}_{\mathrm{PD}}(\overline{\alpha})}$ is defined by $\varphi_c([p]) = \mathsf{c}(\overline{\alpha}, p)$. Broda et al. [3] showed that $\mathscr{A}_{\mathrm{PD}}(\alpha)$ is a quotient of $\overline{\mathscr{A}_{\mathrm{PD}}(\overline{\alpha})}$ by the right-invariant equivalence relation $\equiv_2$ defined on the states of $\overline{\mathscr{A}_{\mathrm{PD}}(\overline{\alpha})}$ by $\beta_1 \equiv_2 \beta_2$ iff $\overline{\beta_1} = \overline{\beta_2}$. Let $\equiv_c$ be the relation $\equiv_2 \circ \equiv_o$. Thus, we have the following result for $\alpha \in \mathrm{RE}(\|)$.

Proposition 7. $\mathscr{A}_{\mathrm{POS}}(\alpha)/{\equiv_c} \simeq (\overline{\mathscr{A}_{\mathrm{PD}}(\overline{\alpha})})/{\equiv_2} \simeq \mathscr{A}_{\mathrm{PD}}(\alpha)$.

References

[1] Valentin Antimirov (1996): *Partial Derivatives of Regular Expressions and Finite Automaton Constructions*. *Theoret. Comput. Sci.* 155(2), pp. 291–319, doi:http://dx.doi.org/10.1016/0304-3975(95)00182-4.

[2] Maurice H. ter Beek, Carlos Martín-Vide & Victor Mitrana (2005): *Synchronized shuffles*. *Theoretical Computer Science* (341), pp. 263–275, doi:10.1016/j.tcs.2005.04.007.

[3] Sabine Broda, António Machiavelo, Nelma Moreira & Rogério Reis (2018): *Automata for regular expressions with shuffle*. *Inf. Comput.* 259(2), pp. 162–173, doi:10.1016/j.ic.2017.08.013.

[4] Sabine Broda, António Machiavelo, Nelma Moreira & Rogério Reis (2021): *Location Based Automata for Expressions with Shuffle*. In Alberto Leporati, Carlos Martín-Vide, Dana Shapira & Claudio Zandron, editors: *Proc. 15th LATA 2021*, *LNCS* 12638, Springer, pp. 43–54, doi:10.1007/978-3-030-68195-1.

[5] Sabine Broda, António Machiavelo, Nelma Moreira & Rogério Reis (2022): *Location Automata for Synchronised Shuffle Expressions*. *Journal of Logical and Algebraic Methods in Programming*. Submitted.

[6] V. M. Glushkov (1961): *The abstract theory of automata*. *Russ. Math. Surv.* 16, pp. 1–53.

[7] Martin Sulzmann & Peter Thiemann (2019): *Derivatives and partial derivatives for regular shuffle expressions*. *J. Comput. Syst. Sci.* 104, pp. 323–341, doi:10.1016/j.jcss.2016.11.010.

Complete Classes of Involutorial Automata

Pál Dömösi

Faculty of Informatics, University of Debrecen
H-4028 Debrecen, Kassai út 26, Hungary

Institute of Mathematics and Informatics, University of Nyíregyháza
H-4400 Nyíregyháza, Sóstói út 31/B, Hungary

domosi@unideb.hu

Miklós Krész

InnoRenew CoE,
SI-6310 Izola, Livade 6, Slovenia

Univerza na Primorskem, Inštitut Andrej Marušič,
SI-6000 Koper, Muzejski trg 2, Slovenia

Juhász Gyula Faculty of Education, University of Szeged,
H-6725 Szeged, Boldogasszony sgt. 6, Hungary

miklos.kresz@innorenew.eu

Adama Diene

Department of Mathematical Science, United Arab Emirates University,
P.O. Box 15551, Al Ain, Abu Dhabi, United Arab Emirates

diene@uaeu.ac.ae

In this paper, for every $i \geq 0$, we characterize the isomorphically complete classes of involutorial automata under the α_i-product. In addition, for every $i \geq 1$, we characterize the homomorphically complete classes of involutorial automata under the α_i-product. The results will be applied for the characterization of the complete classes of soliton automata. The characterization of homomorphically complete classes of involutorial automata under the α_0-product remains as an open problem. Finally, we remark that a very similar characterizations can be given for permutation automata.

1 Introduction

Reversible computation (cf. [1]) plays an important role in the theory of future computer architectures. From automata theoretic point of view involutorial automata or in more general sense permutation automata provide the modelling background for the above field. Soliton automaton [4] based on molecular electronics models as a special involutorial automaton is an example for this approach.

The concept of involutorial automata was introduced by Ito, M. and Duske, J. (1985) [13]. Kutrib, M., Malcher, M., and Schneider, C. referred them as automata with undirected state graphs (2022) [14]. The concepts of Glushkov- product and their isomorphic and homomorphic completeness and the characterization of isomorphic completeness under Glushkov-product are from Glushkov, V. M. (1961) [11]. The homomorphic completeness under the Glushkov- product is characterized by Letichevsky, A.A. (1961) [15]. The concept of α_i-product ($i \geq 0$) is introduced by Gécseg, F. in 1974 (for a review see [9]). The characterizaton of isomorphically complete classes under the α_i-product ($i \geq 0$) is given by Imreh, B. (1977) [12]. The characterization of homomorphically complete classes under the α_0-product is from Dömösi, P. and Ésik, Z. (1988) [5], while the characterization of homomorphically complete classes under the α_1-product is discovered by Ésik, Z. The characterization of homomorphically complete classes under the α_2-product is described by Ésik, Z. (1985) [7]. The characterization of homomorphically complete classes under the α_2-product is also described by Ésik, Z. (1985) [8]. This result implies directly the characterization of homomorphically complete classes under the α_i-product for every integer $i \geq 2$.

In this paper we will consider these like characterizations for the class of involutorial automata. We note that we can give similar characterizations also for the class of permutation automata. Thus, we give the characterization of isomorphically complete classes of involutorial automata (and permutation automata) under the α_i-products for every $i \geq 0$ and also the characterization of homomorphically complete classes of involutorial automata (and permutation automata) under the α_i-products for every $i \geq 1$.

The complete classes of soliton automata as an important application of the above approach will be also characterized by using these results. Though homomorphic representation was already considered in [10], isomorphic representation was not studied and the authors of the above paper followed a different approach: only a special class of soliton automata (strongly deterministic soliton automata) was analyzed.

The characterization of homomorphically complete classes of involutorial automata (and permutation automata) under the α_0-product remains an open problem.

2 Preliminaries

We start with some standard concepts and notation. For all notions and notation not defined here we refer to the monographs [6] and [9]. By an alphabet we mean a finite nonempty set. The elements of an alphabet are called letters. A word over an alphabet Σ is a finite string consisting of letters of Σ. The *length* of a word w, in symbols $|w|$, means the number of letters in w when each letter is counted as many times as it occurs. The string consisting of zero letters is called the *empty word,* written by λ. By definition, $|\lambda| = 0$. The set of all nonempty words over an alphabet Σ will be denoted by Σ^+. In addition, we put $\Sigma^* = \Sigma^+ \cup \{\lambda\}$. Given words $p = x_1 \cdots x_m, q = x_{m+1} \cdots x_n$ for some $x_1, \ldots, x_m, x_{m+1}, \ldots, x_n \in \Sigma$, denote by pq the word $x_1 \cdots x_m x_{m+1} \cdots x_n$. Thus for every $p \in \Sigma^*$ and positive integer n we can define the n-th power p^n of p as the word $w = p_1 \cdots p_n$ with $p_1 = \cdots = p_n = p$. In particular, $p^0 = \lambda$ by definition. Moreover, for every $p \in \Sigma^*$, let $p\lambda = \lambda p = p$. In particular, $\lambda\lambda = \lambda$. By an automaton we mean a finite deterministic automaton without outputs. In other words, by an automaton we mean a system $\mathbf{A} = (A, \Sigma, \delta)$ with a finite nonempty set A of states, a finite nonempty set Σ of inputs, and the transition function $\delta : A \times \Sigma \to A$. We shall use δ in the extended form $\delta : A \times \Sigma^* \to A$ such that for every $a \in A$, let $\delta(a, \lambda) = a$, moreover, for every $a \in A, x \in \Sigma, p \in \Sigma^*$, let $\delta(a, px) = \delta(\delta(a, p), x)$. For every $p \in \Sigma^*$ $\delta_p(a) = \delta(a, p), a \in A$. We shall use the notation $\delta_p, p \in \Sigma^*$ for the mapping $\delta_p : A \to A$ having $\delta_x(a) = \delta(a, x), a \in A$. If for every pair $a, b, \in A$ of states there exists a $p \in \Sigma^+$ with $\delta(a, p) = b$ then we speak about strongly connected automaton. If for every $x \in \Sigma$, δ_x is a permutation of the state set then $\mathscr{A}$ is said to be a permutation automaton. The following statement is obvious.

Proposition **1** Given a permutation automaton $\mathscr{A} = (A, \Sigma, \delta)$, there exists a positive integer n such that for every pair $a \in A, x \in \Sigma$, $\delta(a, x^n) = a$.

$\mathscr{A} = (A, \Sigma, \delta)$ is said to be an involutorial automaton if for every $a \in A, x \in \Sigma$, $\delta(a, x^2) = a$. Obviously, every involutorial automaton is a permutation automaton. Given a finite nonempty set A, $\varphi : A \to A$ is an involutorial mapping over A if $\varphi(\varphi(a)) = a \in \Sigma$. Next we define the class $\mathscr{I}_n = (\{1, \ldots, n\}, \Sigma_{I_n}, \delta_{I_n}), n \geq 1$ of automata such that $\{(\delta_{I_n})_x \mid x \in \Sigma_{I_n}\}$ consists of all involutorial mappings over A. A counter of length $n \geq 1$ is an automaton $\mathscr{C} = (\{1, \ldots, n\}\}, \{x\}, \delta)$ with $\delta(j, x) = j + 1(modn)$ so that $\delta_x : \{1, \ldots, n\} \to \{1, \ldots, n\}$ with $\delta_x(j) = \delta(x, j), j \in \{1, \ldots, n\}$ is a cyclic permutation of $\{1, \ldots, n\}$. The two-state reset automaton is defined as $\mathscr{A}_0 = (\{0, 1\}, \{x_0, x_1\}, \delta_0)$ with $\delta(i, x_j) = j$. Given an automaton $\mathscr{A} = (A, \Sigma, \delta)$, the semigroup $S(\mathscr{A})$ generated by the mappings $\delta_x : A \to A, \delta_x(a) = \delta(a, x), x \in \Sigma$ is called the semigroup of $\mathscr{A}$. Obviously, $S(\mathscr{A})$ is a group if and only if $\mathscr{A}$ is a permutation automaton. We also define the monoid with two right-zero elements as the semigroup $S_0 = \{e, \ell, r\}$ having $ee = e$, $e\ell = \ell e = \ell\ell = r\ell = \ell$ and $er = re = rr = \ell r = r$. We say that an automaton $\mathscr{B}$ homomorphically represents an automaton $\mathscr{A}$ if $\mathscr{B}$ has a subautomaton which can be mapped homomorphically onto $\mathscr{A}$. If $\mathscr{B}$ has a subautomaton which can be mapped isomorphically onto $\mathscr{A}$ then we say that $\mathscr{A}$ can be embedded isomorphically into $\mathscr{B}$, or in other words, $\mathscr{B}$ isomorphically represents $\mathscr{A}$. Similarly, we say that a semigroup S_1 homomorphically represents the semigroup S_2 if S_1 has a subsemigroup which can be mapped homomorphically onto S_2. $\mathscr{A}$ is called autonomous if for every $a \in A, x_1, x_2 \in \Sigma$, $\delta(a, x_1) = \delta(a, x_2)$. Given an automaton $\mathbf{A} = (A, \Sigma, \delta)$, we say that $\mathscr{A}$ satisfies Letichevsky's criterion if there are a state $a \in A$, input letters $x, y \in \Sigma$, input words $p, q \in \Sigma^*$ such that $\delta(a, x) \neq \delta(b, x)$ and $\delta(a, xp) = \delta(a, yq) = a$.

***Proposition* 2** Suppose that an automaton $\mathscr{A}$ has a strongly connected non-autonomous subautomaton. Then $\mathscr{A}$ satisfies Letichevsky's criterion.
Proof. Let $\mathscr{B}$ be a strongly connected non-autonomous subautomaton of $\mathscr{A}$. Because $\mathscr{B}$ is non-autonomous, there are a state $a \in A$, input signs $x, y \in \Sigma$ with $\delta(a, x) \neq \delta(b, x)$. Because $\mathscr{B}$ is strongly connected, there are $p, q \in \Sigma^+$ with $\delta(\delta(a, x), p) = \delta(\delta(a, y), q) = a$. ***QED.***
***Proposition* 3** A permutation automaton $\mathscr{A} = (A, \Sigma, \delta)$ satisfies Letichevsky's criterion if and only if it is non-autonomous.
Proof. First of all, $\mathscr{A}$ is non-autonomous if and only if there are a state $a \in A$, input letters $x, y \in \Sigma$, such that $\delta(a, x) \neq \delta(b, x)$. Therefore, $\mathscr{A}$ does not satisfy Letichevsky's criterion if it is autonomous. Suppose that $\mathscr{A}$ is non-autonomous and let $a \in A$ with $\delta(a, x) \neq \delta(a, y)$ for some $a \in A, x, y \in \Sigma$. Because $\mathscr{A}$ is a permutation automaton, the mappings $\varphi_x : A \to A, \varphi_y : A \to A$ are permutations (over a finite nonvoid set A). Thus, there are positive integers k_1, k_2 such that $(\varphi_x)^{k_1}$ and $(\varphi_y)^{k_2}$ are identical mappings. Therefore, $\delta(a, x^{k_1}) = \delta(a, y^{k_2}) = a$. Put $p = x^{k_1 - 1}, q = y^{k_2 - 1}$. Then $\mathscr{A}$ satisfies Letichevsky's criterion with these $a \in A, x, y \in \Sigma, p, q \in \Sigma^*$. ***QED.***

The next statement is a direct consequence of Proposition 3.
***Proposition* 4** An involutorial automaton satisfies Letichevsky's criterion iff it is non-autonomous.

3 Involutorial automata and α_i-products

Let $\mathscr{A} = (A_t, \Sigma_t, \delta_t), t = 1, \ldots, n$ be a system of automata. Moreover, let Σ be a finite nonvoid set and φ a mapping of $A_1 \times \ldots \times A_n \times \Sigma$ into $\Sigma_1 \times \ldots \Sigma_n \times \Sigma_n$ such that
$\varphi(a_1, \ldots, a_n, x) = (\varphi_1(a_1, \ldots, a_n, x), \ldots, \varphi_n(a_1, \ldots, a_n, x))$
and each $\varphi_j, j = 1, \ldots, n$ is independent of states having indices greater than or equal to $j + i$, where i is a fixed nonnegative integer. We say that the automaton $\mathscr{A} = (A, \Sigma, \delta)$ with $A = A_1 \times \ldots A_n$ and $\delta((a_1, \ldots, a_n), x) = (\delta_1(a_1, \varphi_1(a_1, \ldots, a_n, x)), \ldots, \delta_n(a_n, \varphi_n(a_1, \ldots, a_n, x)))$ is the α_i-product of $\mathscr{A}_t, t = 1,\ldots,n$ with respect to Σ and $\varphi = (\varphi_1, \ldots, \varphi_n)$. For this product we use the shorter notation $\mathscr{A} = \Pi_{t=1}^{n} A_t(\Sigma, \varphi)$. Let $\mathscr{K}$ be a class of automata. We get the concept of Glushkov product [11] omitting the property that each $\varphi_j, j = 1, \ldots, n$ is independent of states having indices greater than or equal to $j + i$. We have the following statement by definition.

***Proposition* 5** Every Glushkov product (and thus, for any $i \geq 0$, every α_i-product) of autonomous automata is autonomous.

By Proposition 4 we can derive as follows.
***Proposition* 6** Let $\mathscr{B}$ be a Glushkov-product (or an α_i-product for some $i \geq 0$) of automata such that none of its component automata has a strongly connected non-autonomous subautomaton having at least two states. Then $\mathscr{B}$ also has no strongly connected non-autonomous subautomaton with two states at least.

We note that an α_0-product of permutation automata is a permutation automaton. Moreover, for any $i \geq 1$, an α_i-product of permutation automata is not necessarily a permutation automaton. Of course, these two observations also hold for α_i-products with a single factor. In addition, an α_0-product with a single factor of an involutorial automaton is also involutorial, but there are α_0-products of involutorial automata with more than one factor such that these products are non-involutorial automata.

Given a class $\mathscr{K}$ of permutation automata, it is called isomorphically complete for the class of permutation automata (or involutorial automata) under the α_i product if any permutation automaton (or involutorial automaton) can be embedded isomorphically into an α_i-product of automata from $\mathscr{K}$. Similarly, the class $\mathscr{K}$ of permutation automata (or involutorial automata) is called homomorphically complete for the class of permutation automata (or involutorial automata) under the α_i product if any permutation automaton (or involutorial automaton) can be represented homomorphically by an α_i-product of automata from $\mathscr{K}$.

By results of B. Imreh [12], we can derive the following two theorems.

***Theorem* 7** The class $\mathscr{K}$ of involutorial automata is isomorphically complete for the class of involutorial automata under the α_0 product if and only if, for every positive integer $n \geq 1$, there exists an automaton $\mathscr{A} \in \mathscr{K}$ such that the automaton $\mathscr{I}_n$ can be embedded isomorphically into an α_0 product of $\mathscr{A}$ with a single factor.

***Theorem* 8** For every $i > 0$, the class $\mathscr{K}$ of involutorial automata is isomorphically complete for the class of involutorial automata under the α_i product if and only if, for every positive integer $n > 0$, there exists an involutorial automaton $\mathscr{A} = (A, \Sigma, \delta) \in \mathscr{K}$ having a subset $\{b_1, \ldots, b_n\} \subseteq A$ of states (consisting of n elements) such that for every $b_i, b_j \in \{b_1, \ldots, b_n\}$ there exists an input sign $x \in \Sigma$ with $\delta(b_i, x) = b_j$.

The next statement can be derived from Theorem 3.36 in [6].

***Theorem* 9** The class $\mathscr{K}$ of involutorial automata is homomorphically complete for the class of involutorial automata under the α_1 product if and only if

(1) Every counter of prime power length can be represented homomorphically by an α_1-product of automata from $\mathscr{K}$,

(2) for every simple group G there is a single factor product $\mathscr{B}$ of an automaton $\mathscr{A} \in \mathscr{K}$ such that S(B) homomorphically represents G.

***Theorem* 10** Given a positive integer $i > 1$, the class $\mathscr{K}$ of involutorial automata is homomorphically complete for the class of involutorial automata under the α_i product if and only if $\mathscr{K}$ contains a non-autonomous involutorial automaton.

Proof. By Proposition 4, an involutorial automaton satisfies Letichevsky's criterion if and only if it is non-autonomous. But then, using Z.Ésik's characterization of homomorphically complete classes under the α_2-product [7], we are ready for $i = 2$. This also shows the sufficiency for $i \geq 2$. On the other hand, the necessity follows from Proposition 6. ***QED.***

The characterization of the homomorphically complete classes of involutorial automata by α_0 products is an open problem.

***Problem* 11** Characterize the homomorphically complete classes of involutorial automata for the class of involutorial automata under the α_0 product.

We note that Theorem 7 can be generalized for the class of permutation automata if we change the terms "involutorial" for "permutation", moreover, instead of $\mathscr{I}_n, n \geq 1$ we consider the automata $\mathscr{P}_n, n \geq 1$ with $\mathscr{P}_n = (\{1, \ldots, n\}, \Sigma_{\mathscr{P}_n}, \delta_{\mathscr{P}_n})$ such that $\{(\delta_{\mathscr{P}_n})_p \mid p \in \Sigma^*_{\mathscr{P}_n}\}$ is the set of all permutations over $\{1, \ldots, n\}$.

Finally, we note that Theorems 8, 9 and 10 can be generalized easily for the class of permutation automata if we change the terms "involutorial" for "permutation".

4 Soliton graphs and soliton automata

In this section we will assume that soliton automata are deterministic, thus we will introduce the concepts with taking into consideration the special structure of deterministic soliton graphs. For a comprehensive review of the topic see [3].

By a graph $G = (V(G), E(G))$ we mean a finite undirected graph in the most general sense, with multiple edges and loops allowed. For detailed background in graph theory see [16]. The *degree* of a vertex v is denoted by $d(v)$, and v is called *external* if $d(v) = 1$ and *internal* if $d(v) \geq 2$. The sets of external and internal vertices of G will be denoted by $Ext(G)$ and $Int(G)$, respectively. *External edges* are those that are incident with at least one external vertex, and an *internal edge* is one that is not external.

A *perfect internal matching* in a graph is a matching that covers all of the internal vertices. An edge $e \in E(G)$ is *allowed (mandatory)* if e is contained in some (respectively, all) perfect internal matching(s) of G. *Forbidden edges* are those that are not allowed. If G is connected and it does not contain forbidden edges, then G is called 1-*extendable*.

A *soliton graph G* is a graph having an external vertex and possessing a perfect internal matching. The set of perfect internal matchings in G are also referred as the *a states* of G. A subgraph G' of G is a *a soliton subgraph* if $Ext(G') = Ext(G) \cap V(G')$. Let G be a soliton graph and M be a state of G, fixed for the rest of this section. An edge $e \in E(G)$ is said to be *M-positive* (*M-negative*) if $e \in M$ (respectively, $e \notin M$). An *M-alternating path (cycle)* in G is a path (respectively, even-length cycle) stepping on M-positive and M-negative edges in an alternating fashion. Let us agree that, if the matching M is understood or irrelevant in a particular context, then it will not be explicitly indicated in these terms. An *external alternating path* is one that has an external endpoint. If both endpoints of the path are external, then it is called a *crossing*. An alternating path is *positive* if it is such at its internal endpoints, meaning that the edges incident with those endpoints are positive.

An internal vertex v of G is called *accessible* from external vertex w in M (or simply v is M-accessible from w), if there exists a positive external M-alternating path connecting w and v. Furthermore, an M-alternating cycle is said to be *M-accessible* from w if some of its vertices is accessible from v in M. It is proved in [4] that impervious edges (no accessible endpoints) does not play role in the soliton transitions (switching operation), hence in the rest of the paper we assume that the soliton graphs do not contain impervious edges.

An *M-alternating unit* is either a crossing or an alternating cycle. *Switching* on an alternating unit γ amounts to changing the sign of each edge along the unit. It is easy to see that the operation of switching on γ creates a new state $S(M, \gamma)$ for G.

Graph G gives rise to a *(deterministic) soliton automaton* $\mathscr{A}(G) = (S(G), X \times X, \delta)$, the set $S(G)$ of states of which consists of the perfect internal matchings of G. The input alphabet $X \times X$ for $\mathscr{A}(G)$ is the set of all (ordered) pairs of external vertices in G – i.e. $X = Ext(G)$ –, and the transition function δ is defined by $\delta(M, (v, w)) = \{S(M, \gamma) | \, \gamma$ is an M-alternating crossing from v to w or $v = w$ and γ is a v-accessible M-alternating cycle$\}$. Nevertheless, if no appropriate M-alternating unit exists from v to w in M, then $\delta(M, (v, w)) = \{M\}$.

5 Complete classes of soliton automata

It was proved in [2] that the transitions of deterministic connected soliton graphs do not contain alternating cycles with the only exception of the so-called chestnuts. A *chestnut* [4] consists of a single cycle of even length with a number of paths leading into it. These paths, however, must obey the restriction that the entry points of different paths have an even distance along the cycle, and the paths can join each other only at even distances from their entries into the cycle. A chestnut is called *trivial* if it has a single external vertex.

The above structural properties are expressed by the following result.

Theorem **12** [2] A soliton graph G is deterministic iff for each component G_i of G, either G_i is a chestnut or G_i does not contain an alternating cycle with respect to any state of G.

Furthermore, it is easy to see by the definition that a transition induced by inputs (v,w) and (w,v) are the same, thus the input pairs can be considered as unordered pairs. Also it is evident that soliton automata are involutorial automata.

With using the preceding observations we first study the α_0-products of deterministic soliton automata. Because of Problem 11, we are considering isomorphic representation only. According to Theorem 7 it can be seen for an isomorphically complete class of soliton automata that this class should contain automata in which distinct states are connected by multiple input signs. From Theorem 12 it can be derived that at most a single state transition can occur between two distinct states in any non-chestnut component of a deterministic soliton automaton. Non-trivial chestnut components however generate multiple input signs, but by induction on the number of non-trivial chestnut components the following can be proved: Consider the subgraph of the state-transition graph induced by the states connected by multiple input signs. This subgraph is a bipartite graph for any soliton automaton by which the conditions of Theorem 7 cannot be satisfied. Therefore we obtained the following result.

Theorem **13** There is no class of deterministic soliton automata which is isomorphically complete for the class of involutorial automata under the α_0-product.

For studying the further members of the product hierarchy,we need a detailed structural analysis of deterministic soliton graphs. Consider first the following construction. Remove each internal forbidden edge belonging to a component of a soliton graph G not being a chestnut and let G' denote the resulted graph. It is clear that each external component (containing external vertex) of G' is either a chestnut or a 1-extendable graph. Then, for each trivial chestnut component G_i add an external edge at the vertex where the single path joining to the cycle of G_i. Furthermore, by Theorem 12, any internal component (containing internal vertices only) of G' consists of a single mandatory edge. Now removing the internal components from G', and applying Theorem 12 for the resulted graph G^*, we can conclude the following.

Theorem **14** Let G be a deterministic soliton graph and $G_1,\ldots,G_k$ are the components of G^*. Then $\mathscr{A}(G)$ is isomorphic with an α_0 product of $\mathscr{A}(G_1),\ldots,\mathscr{A}(G_k)$.

The above result demonstrates that the complete classes of soliton automata should be considered in the frame of 1-extendable graphs and chestnuts. However, by Theorem 9 it is clear that chestnuts will not have role in this context, hence in the followings we will restrict our attention to 1-extendable soliton graphs. Soliton automata based on these graphs will be called *elementary*. For the analysis of deterministic elementary automata, we need a refinement of our structural analysis by a reduction procedure introduced in [2].

A *redex* r in graph G consists of two adjacent edges $e=(u,z)$ and $f=(z,v)$ such that $u\neq v$ are both internal and $d(z)=2$. The vertex z is called the *center* of r, while u and v (e and f) are the two *focal vertices* (respectively, *focal edges*) of r. *Contracting* r in G means creating a new graph G_r from G by deleting the center of r and merging the two focal vertices of r into one vertex s. Another natural simplifying operation on graphs is the removal of an external edge with the property that the degree of its internal endpoint is 2. Such external edges will be called *redundant*.

For an arbitrary soliton graph G, contract all redexes and remove all redundant external edges in an iterative way. By the end of the procedure we obtain a *reduced graph* $r(G)$, i.e. a graph which is free from redexes and redundant external edges.

It can be easily seen that the above method preserves isomorphism; hence, in the light of Theorem 12 and the observations following Theorem 14, it is a central question to characterize the reduced alternating cycle-free 1-extendable soliton graphs. In [2] it was proved that any alternating cycle-free 1-extendable soliton graph reduces to a *generalized tree*, which is a loop-free connected graph not containing even-length cycles. Note that, the odd-length cycles possibly present in a generalized tree must be pairwise edge-disjoint, which explains the terminology.

Summarizing the above observations, complete classes of soliton automata can be characterized by generalized trees. Still, for this goal, we need some preparations.

Recall from [16] that for any $n \in N$, the *n-star* (or simply, *star*) is the bipartite graph $K_{1,n}$ with bipartition (A_1, B_n) such that $|A_1| = 1$, $|B_n| = n$, and the unique vertex of A_1 is adjacent to every vertex of B_n. The above concept is generalized by saying that a tree T with n external vertices ($n \geq 2$) is called an *extended n-star*, if any two vertices of T with degree greater than 2 are at an even distance from each other in T. It can be proved that a tree T with n external vertices ($n \geq 2$) reduces to an n-star iff it is an extended n-star. It is easy to see that an (extended) n-star has n states with a unique transition between any two distinct states.

In the light of Theorem 8, extended n-stars will play a central role in the characterization of isomorphically complete systems. The key point is their embedding into soliton trees, since an approprite mapping from generalized trees to trees can be defined by a further shrinking operation: Considering any generalized tree G, let T_G denote the tree obtained from G by shrinking each cycle of G to a single vertex. Note that the restriction of the states of G are not necessarily states in T_G, thus the cycle-shrinking operation does not preserve isomorphism. Nevertheless, the following relationships hold.

Proposition **15** For a generalized tree G each state of T_G can be extended to a state of G in a unique way. Furthermore, the transitions between distinct states of T_G correspond to the transitions defined by this extension in G. Consequently, $\mathscr{A}(T_G)$ can be isomorphically embedded into an α_1-product of $\mathscr{A}(G)$ with a single factor.

For the characterization of α_i products of soliton automata, the following observation is crucial.

Proposition **16** For a soliton tree T, let T' be a soliton subgraph which is an extended n-star. Then any state of T' can be extended to a state of T in a unique way such that the transitions between distinct states of T' correspond to the transitions defined by this extension in T. Consequently, $\mathscr{A}(T')$ can be isomorphically embedded into an α_1-product of $\mathscr{A}(T)$ with a single factor.

Now we are ready to prove our main result about α_1-products.

Theorem **17** For a class $\mathscr{K}$ of pairwise nonisomorphic elementary soliton automata the following statements are equivalent.

(i) $\mathscr{K}$ is isomorphically complete for the class of involutorial automata under the α_i-product ($i > 0$).

(ii) $\mathscr{K}$ is homomorphically complete for the class of involutorial automata under the α_1-product.

(iii) $\mathscr{K}$ is infinite.

Proof. Because of space restrictions we can provide the basic idea of the proof only. We need to concentrate on the claim "(iii) follows (i)" only as by definitions and Theorem 9 the other directions are clear. For this, using Theorem 8 and the results of this section we need to show that any extended n-star can be isomorphically embedded in a member of $\mathscr{K}$. It can be seen that for a given reduced soliton tree T, there are a finite number of pairwise nonisomorphic reduced generalized tree G with $T = T_G$. Therefore, by Proposition 15 and 16, it is enough to prove that for any $m \geq 1$, there exists a bound $b(m)$ such that any reduced generalized tree with at least $b(m)$ internal vertices contains an extended n-star with m basement vertices (internal vertices incident with external edges) as a soliton subgraph. The proof of this last statement is technically involved with using an induction on m. ***QED.***

Turning to homomorphic representation by α_i-products with $i \geq 2$, we can easily adapt Theorem 10 to soliton automata.

***Theorem* 18** A class $\mathscr{K}$ of deterministic soliton automata is homomorphically complete for the class of involutorial automata under the α_i-product with $i \geq 2$ iff $\mathscr{K}$ contains an automaton $\mathscr{A}(G)$ such that G contains at least 2 external vertices and 2 states.

Finally we note that Theorems 17 and 18 remain true for the class of permutation automata as well.

Acknowledgement

This work was supported by United Arab Emirates Program for Advanced Research (UAEU UPAR) Grant # G00003431.

The second author gratefully acknowledges the European Commission for funding the InnoRenew CoE project (Grant Agreement no. 739574) under the Horizon2020 Widespread-Teaming program and the Republic of Slovenia (Investment funding of the Republic of Slovenia and the European Union of the European Regional Development Fund). He is also grateful for the support of the Slovenian Research Agency (ARRS) through grant N2-0171.

References

[1] B. Aman, G. Ciobanu, R. Glück, R. Kaarsgaard, J. Kari, M. Kutrib, I. Lanese, C. A. Mezzina, L. Mikulski, R. Nagarajan, I. C. C. Phillips, G. M. Pinna, L. Prigioniero, I. Ulidowski & G. Vidal (2020): *Foundations of Reversible Computation*. In: Selected Results of the COST Action IC1405.

[2] M. Bartha & M. Krész (2006): *Deterministic Soliton Graphs*. Informatica (Slovenia) 30, pp. 281–288.

[3] M. Bartha & M. Krész (2010): *Soliton Circuits and Network-Based Automata: Review and Perspectives*. In: Scientific Applications of Language Methods.

[4] J. Dassow & H. Jürgensen (1987): *Soliton Automata*. J. Comput. Syst. Sci. 40, pp. 154–181.

[5] P. Dömösi & Z. Ésik (1988): *Critical Classes for the α_0-Product*. Theor. Comput. Sci. 61, pp. 17–24.

[6] P. Dömösi & C. L. Nehaniv (2005): *Algebraic theory of automata networks - an introduction*. In: SIAM monographs on discrete mathematics and applications.

[7] Z. Ésik (1985): *Homomorphically complete classes of automata with respect to the α_2-product*. Acta Sci. Math. 48, pp. 135–141.

[8] Z. Ésik (1986): *Complete classes of automata for the α_1-product*. Found. Control Eng. 11, pp. 95–107.

[9] F. Gécseg (1986): *Products of Automata*. In: EATCS Monographs in Theoretical Computer Science.

[10] F. Gécseg & H. Jürgensen (1990): *Automata Represented by Products of Soliton Automata*. Theor. Comput. Sci. 74, pp. 163–181.

[11] V. M. Glushkov (1961): *The Abstract Theory of Automata*. Russian Mathematical Surveys 16, pp. 1–53.

[12] B. Imreh (1977): *On α_i-products of automata*. Acta Cybern. 6, pp. 149–162.

[13] M. Ito & J. Duske (1985): *On involutorial automata and involutorial events*. Acta Cybern. 7, pp. 67–79.

[14] M. Kutrib, A. Malcher & C. Schneider (2022): *Finite automata with undirected state graphs*. Acta Informatica 59, pp. 163–181.

[15] A. A. Letichevsky (1961): *Conditions of completeness for finite automata (in Russian)*. Zurn. Vcisl. Matem. i Matem. Fiz. 1, pp. 702–710.

[16] L. Lovász & M. D. Plummer (1986): *Matching Theory*. In: North Holland, Amsterdam.

Rough Grammars, Rough Machines, Rough Languages

Anna Kuczik
Faculty of Informatics, University of Debrecen
Kassai út 26, 4028 Debrecen, Hungary
kuczik.anna@inf.unideb.hu

We define rough grammars and rough finite machines based on similarity relations over the symbols of the alphabet. Our motivation is to study rough languages based on this special case of similarity in order to reduce the complexity of characterizing rough grammars and rough automata.

1 Introduction

Rough sets prove to be useful in many areas of computer science, appearing in several places as a potential option for dealing with questions in which uncertainty occurs.

Uncertainty appears in many areas of life. Let's take an example from the field of medicine, where the issue of uncertainty also appears. There are many symptoms that cannot be used to identify a particular disease without further tests. On the other hand, there are symptoms and combinations of symptoms that make it clear what kind of illness the patient is suffering from. The reason for this is that some symptoms may refer to several problems, so it cannot be determined for sure which problem is causing them at first. In summary, there are combinations of symptoms from which it can be clearly determined that the specific disease is involved, while there are others from which we cannot determine for sure whether the specific disease is the cause, but there is a chance that it is.

An example of this is chicken-pox, a disease which most often appears in children between the ages of 3 and 10. It usually causes epidemics in closed communities in the autumn and winter months. Its initial symptoms are similar to those of a cold or flu: a runny nose, headache, fever, and the most noticeable red rash. Since colds and flu are common in colder periods, it is not possible to determine the disease with these symptoms, but there is a possibility that it is chicken-pox, since these are its primary symptoms. In some cases, the rash appears without any initial symptoms, in which case there is a chance that it may be an allergic rash, but there is still a chance that it is caused by chicken-pox. We can also mention the rashes caused by mites, which spread easily in closed circles, and although they do not have symptoms similar to a cold, they can even be mistaken for similar red, itchy rashes.

We can say that the combination of rashes and cold-like symptoms clearly describes the chicken-pox disease. In addition to cold and flu symptoms, there may be a chance of chicken-pox, further investigations are required to determine it precisely. If red rashes appear without initial symptoms, then there is a chance of chicken-pox, but it is possible that they are caused by something else. There are, therefore, combinations of symptoms from which the disease can be determined with certainty, but also some from which it cannot be determined with certainty. If we want to determine the set of patients infected with chicken-pox based on the symptoms, we can call the part containing certain patients the lower approximation of the set, and the part containing possible patients the upper approximation of the set.

We can speak of set approximation also in the connection with formal languages and automata. For example, we can answer the membership question of regular languages with finite deterministic or nondeterministic automata. In the case of different inputs, the automaton decides whether to accept or not, thus giving a precise answer to the question represented by the given language.

In contrast to traditional automata, rough automata are capable of indicating that they will definitely reach an accepting state, that they may reach an accepting state, or that they will definitely not reach an accepting state for a given input. In this way, rough automata divide the accepted sets into two groups, the definitely accepted and the possibly accepted.

In the following, we would like to examine the question of whether rough automata could be simplified if we are allowed to make assumptions on the relation describing the similarity of elements. The understanding, use and examination of automata with simpler operation becomes easier, so we think that it is important to study the issue of simplification.

To investigate this question, we would like to simplify the automaton and grammar described in articles [1, 2] by using the special case described in [3]. These automata are rough because they have a rough set of states. The first question is, therefore, whether it would not be possible to specify the indiscernible equivalence relation defined on states more simply, so that the equivalence classes contain as few states as possible.

2 Preliminaries

In the following we briefly recall the basics of set approximation and rough sets, see [4, 5]. Our presentation is mostly based on [1, 2].

Definition 1. *A* set approximation space *is a pair* (X,R)*, where* X *is a nonempty set and* R *is an equivalence relation on* X*. If* $x \in X$*, then let* $[x]$ *denote the set* $\{y \in X : xRy\}$*, called an equivalence class or a block under* R*, and let* $X/R = \{[x] : x \in X\}$*. If* xRy*, we may also write* $x \sim_R y$*.*

A set $D \subseteq X$ *is* definable*, if it is a union of equivalence classes, that is,* $D = \bigcup_{x \in D}[x]$*.*

The equivalence relation describes the elements of the set that are similar (or indistinguishable) from a certain point of view we are interested in.

Definition 2. *If* $A \subseteq X$ *in the approximation space* (X,R)*, then* $\underline{A}$ *is the* lower approximation *of* A*, and* $\overline{A}$ *is the* upper approximation *of* A*, which are defined as follows:*

$$\underline{A} = \bigcup\{[x] \in X/R \mid [x] \subseteq A\}, \quad \overline{A} = \bigcup\{[x] \in X/R \mid [x] \cap A \neq \emptyset\},$$

and the pair $(\underline{A}, \overline{A})$ *is a* rough set.

If the elements of an equivalence class are indistinguishable for us, then elements in the lower approximation are those which we can surely classified as belonging to the set. Members of the upper approximation, on the other hand, might be indistinguishable from elements of the set and from elements that do not belong to the set, so they cannot safely be classified as belonging to the set.

Definition 3. *Let* $(\underline{N}, \overline{N})$ *be a rough set in* (N, R_N) *and let* $(\underline{\Sigma}, \overline{\Sigma})$ *be a rough set in* (Σ, R_Σ)*. Let* $Prod \subseteq N \times (N \cup \Sigma)^*$ *with* $(x, \alpha) \in Prod$ *denoted as* $x \to \alpha$*, and let* R_P *be an equivalence relation over* $Prod$*, such that,* pR_Pp' *holds for* $p, p' \in Prod$*, if and only if,* $p : x \to \alpha, p' : x' \to \alpha'$ *such that* xR_Nx' *and* $\alpha = y_1 \dots y_k, \alpha' = y'_1 \dots y'_k$ *with* $y_iR_Ny'_i$*, or* $y_iR_\Sigma y'_i$*,* $1 \leq i \leq k$*.*

Let also for $p^1_i, p^2_j \in Prod, 1 \leq i \leq k, 1 \leq j \leq n$*, and*

$$P_1 = \bigcup_{i=1}^{k}[p^1_i], \quad P_2 = \bigcup_{j=1}^{n}[p^2_j],$$

where square brackets denote the equivalence classes according to R_P.

If $\underline{P_1} = P_1 \cap (\underline{N} \times (\underline{N} \cup \underline{\Sigma})^*)$, $\overline{P_2} = P_2 \cap (\overline{N} \times (\overline{N} \cup \overline{\Sigma})^*)$ and $(\underline{S}, \overline{S})$ is a rough set in (N, R_N), where $\underline{S} \subseteq \underline{N}$, $\overline{S} \subseteq \overline{N}$, then the pair $G = (\underline{G}, \overline{G})$ is called a context-free rough grammar, *where $\underline{G} = (\underline{N}, \underline{\Sigma}, \underline{P_1}, \underline{S})$ and $\overline{G} = (\overline{N}, \overline{\Sigma}, \overline{P_2}, \overline{S})$.*

A rough grammar according to the above definition consists of rough sets of nonterminals and terminals, and a pair of sets of productions. The sets of productions generate sentential forms that consists of symbols from the lower and upper approximations of the alphabets, respectively. We use the notation $\alpha \Rightarrow_G \beta$ to denote the rewriting of the sentential form α to β by using a production of the grammar G, while $\Rightarrow_G^*$ denotes the reflexive and transitive closure of $\Rightarrow_G$.

Rough grammars generate rough languages as follows.

Definition 4. *Let $G = (\underline{G}, \overline{G})$ and let $\underline{L}$, $\overline{L}$ be the languages generated by $\underline{G}$, $\overline{G}$, respectively:*

- $\underline{L} = L(\underline{G}) = \{w \mid w \in \Sigma^*$, *there exists $s \in \underline{S}$ and a derivation $s \Rightarrow_{\underline{G}}^* w\}$,*
- $\overline{L} = L(\overline{G}) = \{w \mid w \in \Sigma^*$, *there existrs $s \in \overline{S}$, and a derivation $s \Rightarrow_{\overline{G}}^* w\}$.*

Then the rough language generated by the rough grammar *G is $L(G) = (\underline{L}, \overline{L})$.*

In [6], rough finite state automata are also defined. Such an automaton consists of a rough set of states and a rough transition function.

Definition 5. *A* rough finite state automaton *is a 4-tuple $M = (Q, R, \Sigma, \delta, I, H)$, where Q is a nonempty finite set, the set of states of M, R is an equivalence relation on Q, Σ is a nonempty finite set (the set of inputs), δ is a rough state transition function, and $I, H \subseteq Q$ are the sets of initial and final states, respectively.*

If $A \subseteq 2^Q$ denotes the definable sets of the approximation space of the set (Q, R), then the state transition function is $\delta : Q \times \Sigma \to A \times A$, where for all $q \in Q$ and $\sigma \in \Sigma$, $\delta(q, \sigma) = (D_1, D_2)$ a rough set, so $\underline{\delta(q, \sigma)} = D_1$, $\overline{\delta(q, \sigma)} = D_2$, $D_1, D_2 \in A$.

We extend the transition function to definable sets of states $\delta^D : A \times \Sigma \to A \times A$ as follows. Let $\delta^D([q], \sigma) = (\underline{\delta^D([q], \sigma)}, \overline{\delta^D([q], \sigma)})$ where

$$\underline{\delta^D([q], \sigma)} = \bigcup_{q' \in [q]} \underline{\delta^D(q', \sigma)}, \quad \overline{\delta^D([q], \sigma)} = \bigcup_{q' \in [q]} \overline{\delta^D(q', \sigma)},$$

and for $B \in A$, $\delta^D(B, \sigma) = (\underline{\delta^D(B, \sigma)}, \overline{\delta^D(B, \sigma)})$ where

$$\underline{\delta^D(B, \sigma)} = \bigcup_{[q] \subseteq B} \underline{\delta^D([q], \sigma)}, \quad \overline{\delta^D(B, \sigma)} = \bigcup_{[q] \subseteq B} \overline{\delta^D([q], \sigma)}.$$

If no confusion arises, we will use δ to denote both δ and δ^D. If we denote by δ^* the extension of transition function to strings (instead of symbols), $\delta^* : A \times \Sigma^* \to A \times A$, then the rough language accepted by the rough automaton can be defined as follows.

Definition 6. *Given a rough finite automaton $M = (Q, \Sigma, R, \delta, I, H)$ as above, consider the following languages:*

- $\underline{L(M)} = \{x \in \Sigma^* \mid \underline{\delta^*(I, x)} \cap H \neq \emptyset\}$,
- $\overline{L(M)} = \{x \in \Sigma^* \mid \overline{\delta^*(I, x)} \cap H \neq \emptyset\}$.

Then the rough language accepted by *M is $L(M) = (\underline{L(M)}, \overline{L(M)})$*

3 Simplification of Rough Grammars and Rough Machines

In this section, we would like to study whether it is possible in certain cases to simplify rough grammar and rough automata. We consider the case examined in [3] where the equivalence relation defined on the words of the language is based on the similarity of letters of the alphabet.

Definition 7. *Let Σ be an alphabet and let R_Σ be an equivalence relation over Σ defining the similarity of letters. For any $a \in \Sigma$, let the set of letters in relation to a (similar to a) be denoted by $\sigma(a) = \{b|b \in \Sigma, (a,b) \in R_\Sigma\}$. For any $w \in \Sigma^*$, let the set of words similar to w be defined as $\sigma(w) = \{a_1^{'},...a_n^{'}|w = a_1...a_n, a_i \in \Sigma, a_i^{'} \in \sigma(a_i), 1 \leq i \leq n\}$.*

Consider the approximation space (Σ^*, R). We say that R is based on the indiscernibility of letters of the alphabet, *if R is defined for $w_1, w_2 \in \Sigma^*$ by $w_1 \sim_R w_2$, if and only if $w_1 \in \sigma(w_2)$ (and $w_2 \in \sigma(w_1)$).*

We will also use the following notions.

Definition 8. *Let S be a finite alphabet, and (S, R_S) an approximation space defined on the set S.*

We say that

- *S is a* crisp set, *if $\underline{S} = \overline{S}$, and*
- *S is a* classical set, *if the number of equivalence classes of R_S is $|[a]| = 1$ for all $a \in S$, so for any $a, b \in S$, the equivalence $a \sim_{R_S} b$ implies $a = b$.*

According to the above definition, S is a crisp set if all equivalence classes in the lower approximation are the same as the equivalence classes in the upper approximation, and vice versa. On the other hand, S is a classical set if no element is indiscernible from another, so no element is in equivalence relation to other elements.

Remark. In the case of conventional sets, relations notation that satisfies the above conditions should be R_{id}. It is clear that if a set is a classical set as described above, it is also a crisp set. Note also that in the approximation spaces (Σ, R_Σ) and (Σ^*, R) defined by R_Σ describing the similarity of the letters of a Σ alphabet as described in [3], the Σ alphabet is a crisp set.

Definition 9. *Let $G = (\underline{G}, \overline{G})$ be a rough grammar, where $\underline{G} = (\underline{N}, \underline{\Sigma}, \underline{P}, \underline{S})$ and $\overline{G} = (\overline{N}, \overline{\Sigma}, \overline{P}, \overline{S})$ and let (N, R_N) be an approximation space interpreted at non-terminals (or (Σ, R_Σ) at terminals).*

We say that, the set of non-terminals (or terminals) in G is a crisp set if N (or Σ) is a crisp set.

We say that, the set of non-terminals (or terminals) in G is a classical set if N (or Σ) is a classical set.

Our first result shows that in the special case mentioned above, the sets of nonterminals of regular rough grammars can be simplified.

Theorem 1. *Let (Σ^*, R) an approximation space interpreted on the strings over the Σ alphabet, where the relation R is based on the indiscernibility of letters of Σ. Let $L = L(G)$ be a language generated by a regular grammar G, and let $((\underline{L}, \overline{L})$ be the rough language approximating $L(G)$ in (Σ^*, R).*

Then a rough grammar $G^{'} = (\underline{G}^{'}, \overline{G^{'}})$ can be constructed, such that $\underline{G}^{'}$ and $\overline{G^{'}}$ are regular grammars, $L(G^{'}) = (\underline{L}, \overline{L})$ and the set of non-terminals in G' is a classical set.

The proof of this statement is based on a construction of G' from any given G, such that the nonterminals of G' are given as a classical set, that is, as a rough set with the identity as the similarity relation. Instead of presenting the proof, we give an example demonstrating the construction.

Example 1. Let $\Sigma = \{a,b,d,e,x,y,z\}$ and let (Σ^*,R) be the approximation space where R is defined by the indiscernibility relation $R_\Sigma : a \sim_{R_\Sigma} b,\ d \sim_{R_\Sigma} e$, and $x \sim_{R_\Sigma} y$, that is, $\Sigma/R_\Sigma = \{[a,b],[d,e],[x,y],[z]\}$. Consider the grammar of $G = (N,\Sigma,P,S)$ where $N = \{S,A,C,D,X,Y,Z\}$ and

$$\begin{aligned} P \;=\; & \{S \to yX, S \to aY, S \to bZ, X \to xA, X \to yD, Y \to zC, Z \to zC, A \to a, A \to b, \\ & C \to d, C \to e, D \to a, D \to b, Y \to a, Y \to b\}. \end{aligned}$$

The generated language is $L(G) = \{aa,ab,ax,axb,aya,ayb,azd,aze,bzd,bze\}$, according to (Σ^*,R) the lower and upper approximations are

$$\begin{aligned} \underline{L(G)} &= \{azd,aze,bzd,bze\}, \\ \overline{L(G)} &= \{aa,ab,ba,bb,axa,axb,bxa,bxb,aya,ayb,bya,byb,azd,aze,bzd,bze\}. \end{aligned}$$

The following sets are the results of the construction from the proof of the previous theorem (the indices are related to the construction algorithm). We group the nonterminals of G into different sets

$$H_{[a],\varepsilon} = \{A,D,Y\},\ H_{[d],\varepsilon} = \{C\},\ H_{[x],[a]} = \{X\},\ H_{[z],[d]} = \{Y,Z\},\ H_{[a],[z]} = \{S\},$$

and based on these we obtain

$$N' = \{S_{[a],[z]}, A_{[a],\varepsilon}, D_{[a],\varepsilon}, Y_{[a],\varepsilon}, C_{[d],\varepsilon}, X_{[x],[a]}, Z_{[z],[d]}, Y_{[z][d]}\},$$

and

$$\begin{aligned} P' \;=\; & \{S_{[a],[z]} \to aX_{[x],[a]},\ S_{[a],[z]} \to aY_{[z],[d]},\ S_{[a],[z]} \to bZ_{[z],[d]},\ X_{[x],[a]} \to xA_{[a],\varepsilon},\ X_{[x],[a]} \to xD_{[a],\varepsilon}, \\ & X_{[x],[a]} \to yD_{[a],\varepsilon},\ X_{[x],[a]} \to yA_{[a],\varepsilon},\ Y_{[z][d]} \to zC_{[d],\varepsilon},\ Z_{[z],[d]} \to zC_{[d],\varepsilon}, A_{[a],\varepsilon} \to a, \\ & A_{[a],\varepsilon} \to b,\ C_{[d],\varepsilon} \to d,\ C_{[d],\varepsilon} \to e,\ D_{[a],\varepsilon} \to a,\ D_{[a],\varepsilon} \to b,\ Y_{[a],\varepsilon} \to a,\ Y_{[a],\varepsilon} \to b, \\ & S_{[a],[z]} \to aY_{[a],\varepsilon},\ Y_{[a],\varepsilon} \to zC_{[d],\varepsilon},\ S_{[a],[z]} \to aZ_{[z],[d]},\ S_{[a],[z]} \to bY_{[z],[d]}\}, \end{aligned}$$

where the lower and upper approximations are

$$\begin{aligned} \underline{P}' \;=\; & \{S_{[a],[z]} \to aY_{[z],[d]},\ S_{[a],[z]} \to bZ_{[z],[d]},\ X_{[x],[a]} \to xA_{[a],\varepsilon},\ X_{[x],[a]} \to xD_{[a],\varepsilon},\ X_{[x],[a]} \to yD_{[a],\varepsilon}, \\ & X_{[x],[a]} \to yA_{[a],\varepsilon},\ Y_{[z][d]} \to zC_{[d],\varepsilon},\ Z_{[z],[d]} \to zC_{[d],\varepsilon},\ A_{[a],\varepsilon} \to a,\ A_{[a],\varepsilon} \to b,\ C_{[d],\varepsilon} \to d, \\ & C_{[d],\varepsilon} \to e,\ D_{[a],\varepsilon} \to a,\ D_{[a],\varepsilon} \to b,\ Y_{[a],\varepsilon} \to a,\ Y_{[a],\varepsilon} \to b,\ Y_{[a],\varepsilon} \to zC_{[d],\varepsilon}, \\ & S_{[a],[z]} \to aZ_{[z],[d]},\ S_{[a],[z]} \to bY_{[z],[d]}\}, \end{aligned}$$

and

$$\begin{aligned} \overline{P}' \;=\; & \{S_{[a],[z]} \to aX_{[x],[a]},\ S_{[a],[z]} \to aY_{[z],[d]},\ S_{[a],[z]} \to bZ_{[z],[d]},\ X_{[x],[a]} \to xA_{[a],\varepsilon},\ X_{[x],[a]} \to xD_{[a],\varepsilon}, \\ & X_{[x],[a]} \to yD_{[a],\varepsilon},\ X_{[x],[a]} \to yA_{[a],\varepsilon},\ Y_{[z][d]} \to zC_{[d],\varepsilon},\ Z_{[z],[d]} \to zC_{[d],\varepsilon},\ A_{[a],\varepsilon} \to a, \\ & A_{[a],\varepsilon} \to b,\ C_{[d],\varepsilon} \to d,\ C_{[d],\varepsilon} \to e,\ D_{[a],\varepsilon} \to a,\ D_{[a],\varepsilon} \to b,\ Y_{[a],\varepsilon} \to a,\ Y_{[a],\varepsilon} \to b, \\ & S_{[a],[z]} \to aY_{[a],\varepsilon},\ Y_{[a],\varepsilon} \to zC_{[d],\varepsilon},\ S_{[a],[z]} \to aZ_{[z],[d]},\ S_{[a],[z]} \to bY_{[z],[d]},\ S_{[a],[z]} \to bX_{[x],[a]}, \\ & S_{[a],[z]} \to bY_{[a],\varepsilon}, \}. \end{aligned}$$

We leave it to the reader to verify that if $G' = (\underline{G}', \overline{G}')$ where $\underline{G}' = (N', \Sigma, \underline{P}', S)$, $\overline{G}' = (N', \Sigma, \overline{P}', S)$, then $L(G') = (\underline{L}, \overline{L})$.

Similarly to nonterminal sets of rough grammars, state sets of rough finite automata can also be simplified, as shown by the following theorem.

Theorem 2. *Let (Σ^*, R) be the approximation space interpreted on strings of the alphabet Σ, where the relation R is based on the indiscernibility of letters of the Σ alphabet, and let $L \subseteq \Sigma^*$ be a regular language.*

Then, a rough finite automaton $M = (Q, R', \Sigma, I, H)$ can be constructed with $L(M) = (\underline{L}, \overline{L})$, such that, $R = R_{id}$, that is, the set Q of states is a classical set, and moreover, $|I| = 1$, the set of initial states is a singleton.

The proof of the theorem is based on a construction of a rough finite automaton from the rough grammar generating the rough regular language with a classical set of nonterminals. Such a grammars exists, according to Theorem 1. The idea is to use the classical construction on the lower and upper approximating production sets to construct the lower and upper approximating transition functions, and combine them in one rough finite state machine.

Now we examine the possibility of simplifying rough finite automata further by having the same lower and upper approximating transition function. In order to still be able to accept the lower and upper approximating languages with one single automaton, we also need to modify the acceptance condition introducing the possibility of accepting different languages with the same machine.

Definition 10. *Let $M = (Q, \Sigma, R, \delta, I, F)$ rough finite state machine. We say that* the state transition relation of M is classical, *if $\underline{\delta(q,a)} = \overline{\delta(q,a)}$ for all $q \in Q$ and $a \in \Sigma$ for (denoted as $\delta(q,a)$), and* $|I| = |\{q_I\}| = 1$.

Then $L(M) = (\underline{L}, \overline{L})$, where

$$\begin{aligned}\underline{L} &= \{w \in \Sigma^* \mid \delta^*(q_I, w) \cap \underline{F} \neq \emptyset \text{ and } \delta^*(q_I, w) \cap \overline{F} \neq \emptyset\},\\ \overline{L} &= \{w \in \Sigma^* \mid \delta^*(q_I, w) \cap \underline{F} = \emptyset \text{ and } \delta^*(q_I, w) \cap \overline{F} \neq \emptyset\}.\end{aligned}$$

Now we can show the following.

Theorem 3. *Let (Σ^*, R) be the approximation space interpreted on strings above the alphabet Σ, where the relation R is based on the indiscernibility of letters of the Σ alphabet, and let $L \subseteq \Sigma^*$ be a regular language.*

Then, a rough finite state machine $M = (Q, R', \Sigma, I, H)$ can be constructed with $L(M) = (\underline{L}, \overline{L})$, such that, the state transition relation of M is classical.

The proof of this statement is similar to the classic construction of finite automata for accepting the intersection of regular languages (also given by finite automata) in the sense that the set of states of the new automaton consists of pairs of states of the ones defining the languages, and the transition relation is given in such a way that the new automaton "follows" both computations simultaneously. In addition, the equivalence relation on the states the new automaton is not the identity, but it is constructed in such a way that together with an appropriately defined set of final states, the lower and upper approximating languages are accepted according to the acceptance criteria given in Definition 10.

References

[1] Basu, S. Rough grammar and rough language. *Foundations of Computing and Decision Sciences*, 28 (2003) 129-140

[2] Basu, S. Rough finite-state automata. *Cybernetics and Systems*, 36:2 (2005) 107-124

[3] Battyányi, P., Mihálydeák, T., Vaszil, Gy. Rough-set-like approximation spaces for formal languages. *Journal of Automata, Languages and Combinatorics*, to appear.

[4] Pawlak, Z. Rough sets. *International Journal of Computer & Information Sciences*, 11 (1982) 341-356

[5] Pawlak, Z. *Rough sets - theoretical aspects of reasoning about data*. Kluwer, Dordrecht, 1991

[6] Sharan, S., Arun, K.S., Tiwari, S.P. Characterizations of rough finite state automata. *International Journal of Machine Learning and Cybernetics*, 8 (2017) 721-730

[7] Sharan, S., Tiwari, S.P. Products of rough finite state machines *arXiv:1210.7575 [math.LO]*, (2012)

Two-Dimensional Typewriter Automata

Taylor J. Smith
Department of Computer Science
St. Francis Xavier University
Antigonish, Nova Scotia, Canada
tjsmith@stfx.ca

A typewriter automaton is a special variant of a two-dimensional automaton that receives two-dimensional words as input and is only capable of moving its input head through its input word in three directions: downward, leftward, and rightward. In addition, downward and leftward moves may only be made via a special “reset" move that simulates the action of a typewriter’s carriage return.

In this paper, we initiate the study of the typewriter automaton model and relate it to similar models, including three-way two-dimensional automata, boustrophedon automata, and returning automata. We study the recognition powers of the typewriter automaton model, establish closure properties of the class of languages recognized by the model, and consider operational state complexity bounds for the specific operation of row concatenation. We also provide a variety of potential future research directions pertaining to the model.

1 Introduction

A two-dimensional typewriter automaton is a variant of the three-way two-dimensional automaton model introduced by Rosenfeld [10]. Similar to the three-way model, the input head of a typewriter automaton can move downward, leftward, and rightward, but the downward and leftward moves work in a fashion similar to the carriage of a typewriter: a special “return" transition repeatedly moves the input head leftward to the leftmost symbol of the current row, then downward by one row. Thus, unlike the three-way two-dimensional automaton model, a typewriter automaton cannot make individual downward or leftward moves.

The typewriter automaton is similar to two other unconventional two-dimensional automaton models that have appeared in the literature [1]: the boustrophedon automaton, which moves its input head from left to right and from right to left on alternating rows of its input word; and the returning automaton, which moves its input head from the leftmost symbol to the rightmost symbol of each row of its input word.

In this paper, we initiate the study of typewriter automata by considering the recognition and closure properties of both deterministic and nondeterministic variants of the model. We show that typewriter automata are capable of recognizing a class of languages “in between" the classes of languages recognized by two-way and three-way two-dimensional automata, and that the class of languages recognized by nondeterministic typewriter automata is equivalent to the class of languages recognized by boustrophedon automata. We further show that the class of languages recognized by typewriter automata is closed under a variety of language operations, and we establish an operational state complexity bound for the specific operation of row concatenation. We conclude by offering a number of potential directions for future research on this model.

2 Preliminaries

A two-dimensional word consists of a finite array, or rectangle, of cells each labelled by a symbol from a finite alphabet Σ. We denote the set of $m \times n$ two-dimensional words over Σ by $\Sigma^{m\times n}$. When a two-dimensional word is written on the input tape of a two-dimensional automaton, the cells around the word are labelled by a special boundary marker $\# \notin \Sigma$. A two-dimensional automaton has a finite state control that is capable of moving its input head in four directions within its input word: up, down, left, and right (denoted by U, D, L, and R, respectively).

Definition 1 (Two-dimensional automaton)**.** A two-dimensional automaton is a tuple $(Q, \Sigma, \delta, q_0, q_{\text{accept}})$, where Q is a finite set of states, Σ is the input alphabet (with $\# \notin \Sigma$ acting as a boundary marker), $\delta : (Q \setminus \{q_{\text{accept}}\}) \times (\Sigma \cup \{\#\}) \to Q \times \{U, D, L, R\}$ is the partial transition function, and $q_0, q_{\text{accept}} \in Q$ are the initial and accepting states, respectively.

We can specify a nondeterministic variant of the model given in Definition 1 by changing the transition function to map to $2^{Q\times\{U,D,L,R\}}$. We denote the deterministic and nondeterministic two-dimensional automaton models by 2DFA-4W and 2NFA-4W, respectively.

By restricting the movement of the input head to prohibit upward movements, we obtain the restricted three-way variant of the two-dimensional automaton model.

Definition 2 (Three-way two-dimensional automaton)**.** A three-way two-dimensional automaton is a tuple $(Q, \Sigma, \delta, q_0, q_{\text{accept}})$ as in Definition 1, where the transition function δ is restricted to use only the directions $\{D, L, R\}$.

We denote deterministic and nondeterministic three-way two-dimensional automata by 2DFA-3W and 2NFA-3W, respectively. Likewise, we may restrict both upward and leftward movements to obtain the two-way two-dimensional automaton model, denoted in the deterministic and nondeterministic cases by 2DFA-2W and 2NFA-2W, respectively.

Additional information about the two-dimensional automaton model can be found in surveys by Inoue and Takanami [6], Kari and Salo [7], and the author [12].

Related to the two-dimensional automaton model is the notion of a boustrophedon automaton, so named because the input head of such an automaton moves from left to right and from right to left on alternating rows of its input word, as in the boustrophedon style of writing. Although the name “boustrophedon automaton" was originally used by Pécuchet in the 1980s in reference to the two-way one-dimensional (string) automaton model [8, 9], the boustrophedon automaton model considered in this paper was introduced by Fernau et al. in 2015 [1] and studied further in subsequent papers [2, 3].

Definition 3 (Boustrophedon automaton)**.** A (two-dimensional) boustrophedon automaton is a tuple $(Q, \Sigma, R, q_0, q_{\text{accept}}, \#, \square)$, where Q, Σ, q_0, and q_{accept} are as in Definition 1, $R \subseteq Q \times (\Sigma \cup \{\#\}) \times Q$ is a finite set of rules, $\# \notin \Sigma$ is a special boundary marker, and $\square$ indicates an erased portion of the input word.

The set of rules R corresponding to a given boustrophedon automaton act as the transition relation of the automaton. The special marker $\square$, which indicates an erased portion of the input word, is used to distinguish symbols that have previously been read by the input head in a given configuration of an automaton. The use of the special marker $\square$ is vital to defining the acceptance criterion for a boustrophedon automaton; since such an automaton must read all of the symbols in its input word before accepting, an $m \times n$ word is accepted if there exists some sequence of configurations ending in an accepting state with $m \times n$ special markers $\square$ on the input tape. A complete description of the boustrophedon automaton model can be found in the papers by Fernau et al. [1, 2, 3]. We denote boustrophedon automata and their deterministic variant by BFA and BDFA, respectively.

$$\begin{tikzcd}
\# \arrow[d] & a_{11} \arrow[l] & a_{12} \arrow[l] & a_{13} \arrow[l] & a_{14} \arrow[l] & a_{15} & \# \\
\# \arrow[r] & a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & \#
\end{tikzcd}$$

Figure 1: An illustration of a typewriter automaton making a reset move $\curvearrowright$ after reading a symbol a_{14}. Each of the moves indicated by arrows is made instantaneously; that is, symbols a_{13}, a_{12}, and a_{11} are not read again as the input head moves to symbol a_{21}.

We now turn to the model that forms the basis of this paper: the typewriter automaton. Like a three-way two-dimensional automaton, a typewriter automaton is only capable of moving its input head in three directions: downward, leftward, and rightward. At the same time, like a boustrophedon automaton, a typewriter automaton reads its input word row-by-row, with the distinction that it moves its input head only from left to right upon entering a given row of its input word. The name of the model is inspired by its behaviour when moving between rows: a typewriter automaton makes use of a special “reset" move, denoted by $\curvearrowright$ and depicted in Figure 1, which simulates the action of a typewriter’s carriage return.

Definition 4 (Typewriter automaton)**.** A (two-dimensional) typewriter automaton is a tuple $(Q, \Sigma, \delta, q_0, q_{\text{accept}})$, where Q, Σ, q_0, and q_{accept} are as in Definition 1, and where the transition function δ is restricted to use only the directions $\{R, \curvearrowright\}$. The reset move $\curvearrowright$ shifts the input head leftward until it reaches the leftmost boundary marker of the current row, then moves the input head one position downward and one position rightward to the first symbol of the subsequent row.

We denote typewriter automata and their deterministic variant by TFA and TDFA, respectively.

Although the typewriter automaton model is quite similar to the three-way two-dimensional automaton model, one notable difference between the two models is that the typewriter automaton cannot make individual downward or leftward input head moves. Once the reset move $\curvearrowright$ is used, the input head moves repeatedly leftward and once downward without reading the symbol underneath. The input head only commences reading symbols again once its input head is positioned over the first symbol of the following row.

We define the notions of a configuration of a typewriter automaton and an accepting computation of a typewriter automaton identically to those used for traditional two-dimensional automata [12]. A configuration of a typewriter automaton $\mathscr{T}$ on an input word $w \in \Sigma^{m \times n}$ is a tuple (q, i, j), where q is the current state of $\mathscr{T}$ and $0 \leq i \leq m+1$ and $0 \leq j \leq n+1$ are the current positions of the input head in w. We then say that $\mathscr{T}$ accepts its input word w if, at some point during its computation, it enters a configuration (q, i, j) where $q = q_{\text{accept}}$.

To simplify the acceptance criterion for typewriter automata, we will not assume that all symbols of the input word must be read. While this choice differs from the definition of acceptance used by boustrophedon automata, it is more in line with the definition used for other two-dimensional automaton models.

Remark. Fernau et al. also studied the so-called returning automaton model, which operates in a manner similar to a boustrophedon automaton, but always moves its input head from the leftmost symbol to the rightmost symbol in a given row [1]. In this sense, a returning automaton is syntactically identical to a boustrophedon automaton, but distinct from a typewriter automaton since typewriter automata are not required to read entire rows of their input words.

3 Recognition Properties

It is well-known that three-way two-dimensional automata recognize strictly fewer languages than the standard (four-way) two-dimensional automaton model [10], and a similar relationship holds between two- and three-way two-dimensional automata. Likewise, all deterministic two-dimensional automaton models recognize strictly fewer languages than their nondeterministic counterparts, resulting in a lattice-like recognition hierarchy between deterministic and nondeterministic two-, three-, and four-way two-dimensional automata [10].

In their original paper, Fernau et al. showed that both deterministic and nondeterministic boustrophedon automata recognize the same class of two-dimensional languages, as they process their input in a manner analogous to a one-dimensional (string) automaton [1]. Moreover, since boustrophedon automata and returning automata are syntactically equivalent, Fernau et al. further showed that both of these models recognize the same class of two-dimensional languages [1]. However, despite their similarities in input head movement, the class of languages recognized by boustrophedon automata is a strict subset of the class of languages recognized by nondeterministic three-way two-dimensional automata [2].

To see where typewriter automata are positioned in the two-dimensional automaton recognition hierarchy, we first compare the model to the weaker two-way two-dimensional automaton. As noted earlier, this variant model is similar to the three-way two-dimensional automaton in Definition 2, but it may only move its input head downward and rightward.

Theorem 5. *Let L_{1s} denote the language of 2×2 words over the alphabet $\Sigma = \{0, 1\}$ having 1s at positions $(1,0)$ and $(0,1)$. Then L_{1s} can be recognized by a typewriter automaton $\mathscr{T}_{1s}$, but not by any two-way two-dimensional automaton.*

Since we do not require nondeterminism to recognize this language, we have that $L_{\mathsf{2DFA\text{-}2W}} \subset L_{\mathsf{TDFA}}$ and $L_{\mathsf{2NFA\text{-}2W}} \subset L_{\mathsf{TFA}}$.

Given our previous observation that typewriter automata and three-way two-dimensional automata are closely related, Theorem 5 may not come as much of a surprise; indeed, we previously noted that $L_{\mathsf{2DFA\text{-}2W}} \subset L_{\mathsf{2DFA\text{-}3W}}$ and $L_{\mathsf{2NFA\text{-}2W}} \subset L_{\mathsf{2NFA\text{-}3W}}$. However, the fact that typewriter automata cannot make independent downward or leftward moves is limiting enough that we can refine the relationships between all three models to show that typewriter automata are positioned between the two- and three-way two-dimensional models.

Theorem 6. *Let L_{stairs} denote the language of $n \times n$ two-dimensional words over the alphabet $\Sigma = \{0, 1\}$ having a contiguous[1] path of 1s forming a staircase pattern from the top-left corner to the bottom-right corner. Then L_{stairs} can be recognized by a three-way two-dimensional automaton $\mathscr{A}$, but not by any typewriter automaton.*

Again, since we do not require nondeterminism to recognize this language, we have that $L_{\mathsf{TDFA}} \subset L_{\mathsf{2DFA\text{-}3W}}$ and $L_{\mathsf{TFA}} \subset L_{\mathsf{2NFA\text{-}3W}}$.

Similar to the more standard two-dimensional automaton models, we can show that a separation exists between the deterministic and nondeterministic variants of the typewriter automaton model with another small example language.

Theorem 7. *Let L_L denote the language of $m \times n$ two-dimensional words over the alphabet $\Sigma = \{0, 1\}$ where $m, n \geq 2$, the first column and last row of each word consist entirely of 1s, and all other symbols are 0. Then L_L can be recognized by a nondeterministic typewriter automaton $\mathscr{T}_L$, but not by any deterministic typewriter automaton.*

[1] By contiguous, we mean that pairs of 1s are adjacent either horizontally or vertically, but not diagonally.

$$\begin{tikzcd}
 & L_{\mathsf{2NFA\text{-}4W}} & & \\
L_{\mathsf{2DFA\text{-}4W}} \arrow[ur] & & & \\
 & L_{\mathsf{2NFA\text{-}3W}} \arrow[uu] & & \\
L_{\mathsf{2DFA\text{-}3W}} \arrow[uu] \arrow[ur] & & & \\
 & L_{\mathsf{TFA}} \arrow[uu] \arrow[r, phantom, "="] & L_{\mathsf{BFA}} \arrow[uul] \arrow[r, phantom, "="] & L_{\mathsf{BDFA}} \\
L_{\mathsf{TDFA}} \arrow[uu] \arrow[ur] & & & \\
 & L_{\mathsf{2NFA\text{-}2W}} \arrow[uu] & & \\
L_{\mathsf{2DFA\text{-}2W}} \arrow[uu] \arrow[ur] & & &
\end{tikzcd}$$

Figure 2: Inclusions among classes of languages recognized by two-dimensional automaton models mentioned in this paper. An arrow $L_{\mathsf{A}} \rightarrow L_{\mathsf{B}}$ indicates the relation $L_{\mathsf{A}} \subset L_{\mathsf{B}}$.

It is known that a nondeterministic boustrophedon automaton can recognize the same language L_{L} defined in Theorem 7 and, therefore, so too can a deterministic boustrophedon automaton [1]. More generally, given the similarities between the input head movement of a typewriter automaton and that of a returning automaton, and the equivalence in recognition power between returning automata and boustrophedon automata, we can establish the following result.

Theorem 8. *Nondeterministic typewriter automata and boustrophedon automata recognize the same class of languages.*

By combining Theorem 8 with the observation that $L_{\mathsf{TDFA}} \subset L_{\mathsf{TFA}}$, we see that boustrophedon automata can recognize all languages recognized by deterministic typewriter automata. However, the deterministic version of Theorem 8 does not hold, since (for example) the language L_{L} is recognized by a boustrophedon automaton but not by a deterministic typewriter automaton.

Altogether, the results established in this section produce the inclusion hierarchy depicted in Figure 2.

4 Closure Properties

The closure properties of three-way two-dimensional automata are well-known. In the nondeterministic case, closure holds for the operations of union, row concatenation, and reversal (or row reflection) [4, 5, 16, 17]. The operations of row concatenation and reversal are depicted in Figure 3. However, the model is not closed under the operations of intersection, complement, column concatenation, or rotation [5]. Focusing on the deterministic model, on the other hand, closure holds only for the operation of complement [17].

For boustrophedon automata, it is known that the class of languages recognized by this model is closed under union, intersection, complement, row concatenation, and reversal [1, 2, 11], but not under column concatenation or rotation [2, 11].

Returning to the two-way two-dimensional automaton model briefly mentioned earlier, it is known that the deterministic variant is closed under complement while the nondeterministic variant is closed under union; closure does not hold for any other operation [13, 15].

$$v \ominus w = \begin{array}{cccccc} \# & \# & & & \# & \# \\ \# & v_{1,1} & \cdots & v_{1,n} & \# \\ & \vdots & \ddots & \vdots & \\ \# & v_{m,1} & \cdots & v_{m,n} & \# \\ \# & w_{1,1} & \cdots & w_{1,n} & \# \\ & \vdots & \ddots & \vdots & \\ \# & w_{m',1} & \cdots & w_{m',n} & \# \\ \# & \# & & \# & \# \end{array}$$

(a) Row concatenation of two words v and w

$$w^{\mathrm{R}} = \begin{array}{ccccc} \# & \# & & \# & \# \\ \# & w_{m,1} & \cdots & w_{m,n} & \# \\ & \vdots & \ddots & \vdots & \\ \# & w_{1,1} & \cdots & w_{1,n} & \# \\ \# & \# & & \# & \# \end{array}$$

(b) Reversal of a word w

Figure 3: Illustrations of the operations of row concatenation and reversal

Since typewriter automata are positioned between the two- and three-way two-dimensional automaton models in the recognition hierarchy, we can establish closure results for the typewriter automaton model using proofs analogous to those used for other models.

Proposition 9. *L_{TFA} is closed under union. L_{TDFA} is closed under complement.*

Moreover, although it is possible to prove additional closure properties directly, we obtain closure for some other operations applied to typewriter automaton languages as a consequence of Theorem 8.

Theorem 10. *L_{TFA} is closed under intersection, complement, row concatenation, and reversal.*

In a sense, Theorem 10 reveals more about the typewriter automaton model than Proposition 9, as neither the nondeterministic two- nor three-way two-dimensional automaton models are closed under intersection or complement, and only the nondeterministic three-way model is closed under row concatenation and reversal.

4.1 Operational State Complexity

Although state complexity is a well-studied measure for traditional automaton models, the same cannot be said for two-dimensional automaton models. The primary barrier to obtaining such results is the fact that there are currently no known general techniques for proving state complexity lower bounds in two dimensions, in contrast to the techniques known for one-dimensional (string) automata.

That being said, one-dimensional state complexity techniques have been used successfully to obtain bounds for projections of two-dimensional languages [14], and a number of elementary operational state complexity upper bounds have been obtained directly for certain two-dimensional language operations [13]. Of these elementary bounds, one pertaining to the three-way two-dimensional automaton model is particularly relevant:

Proposition 11. *Given two nondeterministic three-way two-dimensional automata $\mathscr{A}$ and $\mathscr{B}$ with m and n states, respectively, the automaton $\mathscr{A} \ominus \mathscr{B}$ contains at most $m+n+2$ states.*

The idea underlying the proof of Proposition 11 is that an additional two states are needed to move the input head from its accepting position in the input word to $\mathscr{A}$ to its initial position in the input word to $\mathscr{B}$.

Since the operation of row concatenation is closed for nondeterministic typewriter automata, we can investigate its state complexity. Due to the fact that typewriter automata use a single reset move to shift the input head leftward and downward, we immediately incur a savings in the number of states needed to recognize a row concatenation language.

Theorem 12. *Given two nondeterministic typewriter automata $\mathscr{T}_1$ and $\mathscr{T}_2$ with m and n states, respectively, the automaton $\mathscr{T}_1 \ominus \mathscr{T}_2$ contains at most $m+n$ states.*

Note that $m+n$ states is presumably the matching lower bound for the row concatenation of two typewriter automaton languages, since the individual automata require m and n states to recognize their respective languages. However, as mentioned before, there is no known general technique for establishing lower bounds for two-dimensional automaton models.

5 Conclusion

In this paper, we introduced the notion of a typewriter automaton, drew connections between this model and the existing models of boustrophedon and returning automata, established recognition properties of the model relative to other two-dimensional automaton models, showed closure properties of certain language operations, and considered operational state complexity bounds. We showed that a typewriter automaton is capable of recognizing languages in between the two- and three-way variants of the standard two-dimensional automaton model, while the nondeterministic typewriter automaton has a recognition power equivalent to that of both boustrophedon and returning automata. We further established closure of the union, row concatenation, and reversal language operations for typewriter automata, and of the complement operation for the deterministic variant. Lastly, we investigated the operational state complexity of row concatenation for nondeterministic typewriter automata.

A number of fruitful questions remain surrounding the typewriter automaton model. For instance, having established which operations are closed for both deterministic and nondeterministic typewriter automata, it would be interesting to establish state complexity bounds for these operations, possibly using some as-yet-unknown proof technique. Beyond settling the question for typewriter automata, the same techniques may be applicable to boustrophedon automata or to the more general three-way two-dimensional automaton model. Future research may also consider the computational complexity of certain decision problems for typewriter automata, as has been done for boustrophedon automata.

In one of their papers on boustrophedon automata [2], Fernau et al. mention that boustrophedon scanning is not a new innovation, and that the technique is used widely in image processing applications; for example, in converting two-dimensional data to a one-dimensional string for compression, or in preprocessing data for optical character recognition. As the action of a typewriter automaton is somewhat similar to that of a boustrophedon (or a returning) automaton, it is not unreasonable to assume that the typewriter automaton model might also find applications in image processing or recognition. In particular, the left-to-right movement of the input head combined with access to the reset move may be suitable for applications such as edge finding or mapping contiguous segments in images represented as two-dimensional words. Alongside the restricted two- and three-way two-dimensional automaton models, typewriter automata present a conceptually simpler model of computation relative to the standard four-way two-dimensional automaton model.

Acknowledgements

The author would like to thank Ian McQuillan for suggesting the idea of a “two-dimensional typewriter automaton" during his PhD defence in August 2021.

References

[1] H. Fernau, M. Paramasivan, M. L. Schmid & D. G. Thomas (2015): *Scanning Pictures the Boustrophedon Way*. In R. P. Barneva, B. B. Bhattacharya & V. E. Brimkov, editors: *Proc. of IWCIA 2015*, *LNCS* 9448, Springer, Berlin, pp. 202–216, doi:10.1007/978-3-319-26145-4_15.

[2] H. Fernau, M. Paramasivan, M. L. Schmid & D. G. Thomas (2018): *Simple picture processing based on finite automata and regular grammars*. *J. Comput. System Sci.* 95, pp. 232–258, doi:10.1016/j.jcss.2017.07.011.

[3] H. Fernau, M. Paramasivan & D. G. Thomas (2016): *Regular Array Grammars and Boustrophedon Finite Automata*. In H. Bordihn, R. Freund, B. Nagy & G. Vaszil, editors: *Short Papers of NCMA 2016*, Institut für Computersprachen, TU Wien, Vienna, pp. 55–63. Available at `https://www.cs.uni-potsdam.de/NCMA/ShortPapers/NCMA2016ShortPapers.pdf`.

[4] K. Inoue & I. Takanami (1979): *Closure Properties of Three-Way and Four-Way Tape-Bounded Two-Dimensional Turing Machines*. *Inform. Sci.* 18(3), pp. 247–265, doi:10.1016/0020-0255(79)90048-3.

[5] K. Inoue & I. Takanami (1979): *Three-Way Tape-Bounded Two-Dimensional Turing Machines*. *Inform. Sci.* 17, pp. 195–220, doi:10.1016/0020-0255(79)90017-3.

[6] K. Inoue & I. Takanami (1991): *A Survey of Two-Dimensional Automata Theory*. *Inform. Sci.* 55(1–3), pp. 99–121, doi:10.1016/0020-0255(91)90008-I.

[7] J. Kari & V. Salo (2011): *A Survey on Picture-Walking Automata*. In W. Kuich & G. Rahonis, editors: *Algebraic Foundations in Computer Science*, *LNCS* 7020, Springer, Berlin, pp. 183–213, doi:10.1007/978-3-642-24897-9_9.

[8] J.-P. Pécuchet (1985): *Automates Boustrophédon et Mots Infinis*. *Theoret. Comput. Sci.* 35, pp. 115–122, doi:10.1016/0304-3975(85)90009-X.

[9] J.-P. Pécuchet (1985): *Automates Boustrophédon, Semi-Groupe de Birget et Monoïde Inversif Libre*. *RAIRO Theor. Inform. Appl.* 19(1), pp. 71–100, doi:10.1051/ita/1985190100711.

[10] Azriel Rosenfeld (1979): *Picture Languages: Formal Models for Picture Recognition*. Computer Science and Applied Mathematics, Academic Press, New York.

[11] G. Siromoney, R. Siromoney & K. Krithivasan (1973): *Picture Languages with Array Rewriting Rules*. *Inform. and Control* 22(5), pp. 447–470, doi:10.1016/S0019-9958(73)90573-1.

[12] T. J. Smith (2019): *Two-Dimensional Automata*. Technical report 2019-637, Queen's University, Kingston. Available at `http://research.cs.queensu.ca/TechReports/Reports/2019-637.pdf`.

[13] T. J. Smith (2021): *Closure, Decidability, and Complexity Results for Restricted Variants of Two-Dimensional Automata*. PhD thesis, Queen's University. Available at `http://hdl.handle.net/1974/29180`.

[14] T. J. Smith & K. Salomaa (2020): *Recognition and Complexity Results for Projection Languages of Two-Dimensional Automata*. In G. Jirásková & G. Pighizzini, editors: *Proc. of DCFS 2020*, *LNCS* 12442, Springer, Berlin, pp. 206–218, doi:10.1007/978-3-030-62536-8_17.

[15] T. J. Smith & K. Salomaa (2021): *Concatenation Operations and Restricted Variants of Two-Dimensional Automata*. In T. Bureš et al., editors: *Proc. of SOFSEM 2021*, *LNCS* 12607, Springer, Berlin, pp. 147–158, doi:10.1007/978-3-030-67731-2_11.

[16] A. Szepietowski (1989): *On Three-Way Two-Dimensional Turing Machines*. *Inform. Sci.* 47(2), pp. 135–147, doi:10.1016/0020-0255(89)90010-8.

[17] A. Szepietowski (1992): *Some Remarks on Two-Dimensional Finite Automata*. *Inform. Sci.* 63(1–2), pp. 183–189, doi:10.1016/0020-0255(92)90068-J.

Author Index