EPTCS 422

Proceedings of the

15th International Workshop on

Non-Classical Models of Automata and Applications

Loughborough University, July 21-22, 2025

Edited by: Nelma Moreira and Luca Prigioniero

Published: 19th July 2025
DOI: 10.4204/EPTCS.422
ISSN: 2075-2180
Open Publishing Association

Preface

The 15th International Workshop on Non-Classical Models of Automata and Applications (NCMA 2025) was held in Loughborough, UK, on July 21 and 22, 2025, organized by the Department of Computer Science at Loughborough University and co-located with the 26th International Conference on Descriptional Complexity of Formal Systems (DCFS 2025, 22-24 July).

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 various 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 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. After that, the series continued in Debrecen, Hungary (2022), Famagusta, North Cyprus (2023), and Göttingen, Germany (2024).

The invited lectures at NCMA 2025 were the following:

- Laure Daviaud (University of East Anglia, Norwich, UK): Weighted Automata and Cost Register Automata: (Un)Decidability
- Ian McQuillan (University of Saskatchewan, Saskatoon (SK), Canada): Inductive Inference of Lindenmayer Systems: Computational and Descriptional Complexity (joint invited lecture with DCFS 2025)

The 7 regular contributions were selected out of 13 submissions by a total of 26 authors from 9 different countries by the following members of the Program Committee:

- Marcella Anselmo (Salerno, Italy)
- Sabine Broda (Porto, Portugal)
- Cezar Câmpeanu (Charlottetown (PEI), Canada)
- Joel Day (Loughborough, UK)
- Frank Drewes (Umeå, Sweden)
- Zsolt Gazdag (Szeged, Hungary)
- Bruno Guillon (Clermont-Ferrand, France)
- Zbyněk Křivka (Brno, Czechia)
- Carlo Mereghetti (Milan, Italy)
- Nelma Moreira (Porto, Portugal, co-chair)
- František Mráz (Prague, Czechia)

N. Moreira and L. Prigioniero (Eds.): 15th International Workshop on Non-Classical Models of Automata and Applications (NCMA 2025).
EPTCS 422, 2025, pp. i–ii, doi:10.4204/EPTCS.422.0

- Benedek Nagy (Famagusta, North Cyprus)
- Luca Prigioniero (Loughborough, UK, co-chair)
- Juraj Sebej (Kosice, Slovakia)
- Hellis Tamm (Tallin, Estonia)
- Stefan Siemer (Göttingen, Germany)
- Matthias Wendlandt (Giessen, Germany)

The following additional reviewers helped in the evaluation process:

- Silvio Capobianco
- Flavio D'Alessandro
- Guilherme Duarte
- Szilárd Zsolt Fazekas
- Jarkko Kari
- Martin Kutrib
- Hendrik Maarand
- Giovanni Pighizzini
- Priscilla Raucci
- Rocco Zaccagnino

In addition to the invited presentations and the regular contributions, NCMA 2025 featured 4 informal presentations to emphasize its workshop character.

A special issue of RAIRO - Theoretical Informatics and Applications containing extended versions of selected contributions to NCMA 2025 will also be edited after the workshop. The extended papers will undergo the standard refereeing process of the journal.

We are grateful to the two invited speakers, all authors who submitted a paper to NCMA 2025, the members of the Program Committee, and their sub-reviewers who helped evaluating the submissions. We are deeply indebted to Robert Mercaş from the Department of Computer Science of Loughborough University for his outstanding efforts in the local organization of the workshop. We also greatly appreciated the financial support of the Center of Mathematics of the University of Porto (CMUP).

July 2025

Nelma Moreira
Luca Prigioniero

Table of Contents

Idefix-Closed Languages and Their Application in Contextual Grammars

Marvin Ködding
Institut für Mathematik und Informatik, Pädagogische Hochschule Heidelberg
Im Neuenheimer Feld 561, 69120 Heidelberg, Germany
koedding@ph-heidelberg.de

Bianca Truthe
Institut für Informatik, Universität Giessen
Arndtstr. 2, 35392 Giessen, Germany
bianca.truthe@informatik.uni-giessen.de

In this paper, we continue the research on the power of contextual grammars with selection languages from subfamilies of the family of regular languages. We investigate infix-, prefix-, and suffix-closed languages (referred to as idefix-closed languages) and compare such language families to some other subregular families of languages (finite, monoidal, nilpotent, combinational, (symmetric) definite, ordered, non-counting, power-separating, commutative, circular, union-free, star, and comet languages). Further, we compare the families of the hierarchies obtained for external and internal contextual grammars with the language families defined by these new types for the selection. In this way, we extend the existing hierarchies by new language families. Moreover, we solve an open problem regarding internal contextual grammars with suffix-closed selection languages.

1 Introduction

Contextual grammars, first proposed by Solomon Marcus [19] provide a formal framework for modeling the generation of natural languages. In this model, derivations proceed by adjoining pairs of 'contexts' – that is, ordered pairs of words (u,v) – to existing well-formed sentences. Specifically, an external application of a context (u,v) to a word x yields the word uxv, whereas an internal application produces every word of the form $x_1ux_2vx_3$ for which $x_1x_2x_3 = x$. To regulate the derivational process, each context is associated with a 'selection' language: a context (u,v) may only be applied around a word x if x belongs to its designated selection language. By constraining selection languages to belong to a prescribed family F, one obtains contextual grammars with selection in F.

The initial investigations into external contextual grammars with regular selection languages were conducted by Jürgen Dassow [6] and were subsequently extended – both for external and internal variants by Jürgen Dassow, Florin Manea, and Bianca Truthe [8, 9]. These studies examined the impact of various subregular restrictions on selection languages. In the present work, we further refine this hierarchy by introducing families of 'idefix-closed' subregular languages and explore the generative power of both external and internal contextual grammars whose selection languages lie in these newly defined families. With 'idefix-closed', we mean prefix-, suffix-, or infix-closed.

Especially, in the present paper, we solve an open problem regarding internal contextual grammars with suffix-closed selection languages which was raised already several years ago in [30].

N. Moreira and L. Prigioniero (Eds.): 15th International Workshop on Non-Classical Models of Automata and Applications (NCMA 2025).
EPTCS 422, 2025, pp. 1–14, doi:10.4204/EPTCS.422.1

2 Preliminaries

Throughout the paper, we assume that the reader is familiar with the basic concepts of the theory of automata and formal languages. For details, we refer to [25]. Here we only recall some notation, definitions, and previous results which we need for the present research.

An alphabet is a non-empty finite set of symbols. For an alphabet V, we denote by V^* and V^+ the set of all words and the set of all non-empty words over V, respectively. The empty word is denoted by λ. For a word w and a letter a, we denote the length of w by $|w|$ and the number of occurrences of the letter a in the word w by $|w|_a$. For a set A, we denote its cardinality by $|A|$.

The family of the regular languages is denoted by *REG*. Any subfamily of this set is called a subregular language family.

For a language L over an alphabet V, we set

$$\begin{aligned} \mathit{Inf}(L) &= \{\, y \mid xyz \in L \text{ for some } x,z \in V^* \,\}, \\ \mathit{Pre}(L) &= \{\, x \mid xy \in L \text{ for some } y \in V^* \,\}, \\ \mathit{Suf}(L) &= \{\, y \mid xy \in L \text{ for some } x \in V^* \,\} \end{aligned}$$

as the infix-, prefix-, and suffix-closure of L, respectively. If the language L is regular, then also $\mathit{Inf}(L)$ and $\mathit{Suf}(L)$ are regular.

2.1 Some Subregular Language Families

We consider the following restrictions for regular languages. In the following list of properties, we give already the abbreviation which denotes the family of all languages with the respective property. Let L be a regular language over an alphabet V. With respect to the alphabet V, the language L is said to be

- *monoidal* (*MON*) if and only if $L = V^*$,
- *nilpotent* (*NIL*) if and only if it is finite or its complement $V^* \setminus L$ is finite,
- *combinational* (*COMB*) if and only if it has the form $L = V^*X$ for some subset $X \subseteq V$,
- *definite* (*DEF*) if and only if it can be represented in the form $L = A \cup V^*B$ where A and B are finite subsets of V^*,
- *symmetric definite* (*SYDEF*) if and only if $L = EV^*H$ for some regular languages E and H,
- *infix-closed* (*INF*) if and only if, for any three words over V, say $x \in V^*$, $y \in V^*$ and $z \in V^*$, the relation $xyz \in L$ implies the relation $y \in L$,
- *prefix-closed* (*PRE*) if and only if, for any two words over V, say $x \in V^*$ and $y \in V^*$, the relation $xy \in L$ implies the relation $x \in L$,
- *suffix-closed* (*SUF*) if and only if, for any two words over V, say $x \in V^*$ and $y \in V^*$, the relation $xy \in L$ implies the relation $y \in L$,
- *ordered* (*ORD*) if and only if the language is accepted by some deterministic finite automaton

  $$\mathcal{A} = (V, Z, z_0, F, \delta)$$

  with an input alphabet V, a finite set Z of states, a start state $z_0 \in Z$, a set $F \subseteq Z$ of accepting states and a transition mapping δ where $(Z, \preceq)$ is a totally ordered set and, for any input symbol $a \in V$, the relation $z \preceq z'$ implies $\delta(z,a) \preceq \delta(z',a)$,

- *commutative* (*COMM*) if and only if it contains with each word also all permutations of this word,
- *circular* (*CIRC*) if and only if it contains with each word also all circular shifts of this word,
- *non-counting* (*NC*) if and only if there is a natural number $k \geq 1$ such that, for any three words $x \in V^*$, $y \in V^*$, and $z \in V^*$, it holds $xy^k z \in L$ if and only if $xy^{k+1}z \in L$,
- *star-free* (*SF*) if and only if L can be described by a regular expression which is built by concatenation, union, and complementation,
- *power-separating* (*PS*) if and only if, there is a natural number $m \geq 1$ such that for any word $x \in V^*$, either $J_x^m \cap L = \emptyset$ or $J_x^m \subseteq L$ where $J_x^m = \{ x^n \mid n \geq m \}$,
- *union-free* (*UF*) if and only if L can be described by a regular expression which is only built by concatenation and Kleene closure,
- *star* (*STAR*) if and only if $L = H^*$ for some regular language $H \subseteq V^*$,
- *left-sided comet* (*LCOM*) if and only if $L = EG^*$ for some regular language E and a regular language $G \notin \{\emptyset, \{\lambda\}\}$,
- *right-sided comet* (*RCOM*) if and only if $L = G^*H$ for some regular language H and a regular language $G \notin \{\emptyset, \{\lambda\}\}$,
- *two-sided comet* (*2COM*) if and only if $L = EG^*H$ for two regular languages E and H and a regular language $G \notin \{\emptyset, \{\lambda\}\}$.

We group the language families *SUF*, *PRE* and *INF* under the term idefix-closed families. We remark that monoidal, nilpotent, combinational, (symmetric) definite, ordered, star-free, union-free, star, and (left-, right-, or two-sided) comet languages are regular, whereas non-regular languages of the other types mentioned above exist. Here, we consider among the suffix-closed, commutative, circular, non-counting, and power-separating languages only those which are also regular. By *FIN*, we denote the family of languages with finitely many words. In [20], it was shown that the families of the regular non-counting languages and the star-free languages are equivalent ($NC = SF$).

Some properties of the languages of the classes mentioned above can be found in [26] (monoids), [11] (nilpotent languages), [14] (combinational and commutative languages), [24] (definite languages), [23] (symmetric definite languages), [5] (prefix-closed languages), [12] and [5] (suffix-closed languages), [27] (ordered languages), [18] (circular languages), [20] (non-counting and star free languages), [28] (power-separating languages), [2] (union-free languages), [3] (star languages), [4] (comet languages).

2.2 Contextual Grammars

Let $\mathcal{F}$ be a family of languages. A contextual grammar with selection in $\mathcal{F}$ is a triple $G = (V, \mathcal{S}, A)$ with the following components:

- V is an alphabet.
- $\mathcal{S}$ is a finite set of selection pairs (S, C) where S is called selection language and C is a set of so-called contexts. Any selection language S is a language in the family $\mathcal{F}$ with respect to an alphabet $U \subseteq V$. Any set C of contexts is a finite set $C \subset V^* \times V^*$ where, for each context $(u, v) \in C$, at least one side is not empty: $uv \neq \lambda$.
- A is a finite subset of V^* (its elements are called axioms).

We write a selection pair (S,C) also as $S \to C$. In the case that C is a singleton set $C = \{(u,v)\}$, we also write $S \to (u,v)$.

For a contextual grammar $G = (V, \{S_1 \to C_1, S_2 \to C_2, \ldots, S_n \to C_n\}, A)$, we set

$$\ell_{\mathrm{A}}(G) = \max\{\, |w| \mid w \in A \,\},\ \ell_{\mathrm{C}}(G) = \max\{\, |uv| \mid (u,v) \in C_i, 1 \le i \le n \,\},\ \ell(G) = \ell_{\mathrm{A}}(G) + \ell_{\mathrm{C}}(G) + 1.$$

We now define the derivation modes for contextual grammars with selection.

Let $G = (V, \mathcal{S}, A)$ be a contextual grammar with selection. A direct external derivation step in G is defined as follows: a word x derives a word y (written as $x \Longrightarrow_{\mathrm{ex}} y$) if and only if there is a pair $(S,C) \in \mathcal{S}$ such that $x \in S$ and $y = uxv$ for some pair $(u,v) \in C$. Intuitively, one can only wrap a context $(u,v) \in C$ around a word x if x belongs to the corresponding selection language S.

A direct internal derivation step in G is defined as follows: a word x derives a word y (written as $x \Longrightarrow_{\mathrm{in}} y$) if and only if there are words x_1, x_2, x_3 with $x_1x_2x_3 = x$ and there is a selection pair $(S,C) \in \mathcal{S}$ such that $x_2 \in S$ and $y = x_1ux_2vx_3$ for some pair $(u,v) \in C$. Intuitively, we can only wrap a context $(u,v) \in C$ around a subword x_2 of x if x_2 belongs to the corresponding selection language S.

By $\Longrightarrow_{\mu}^{*}$ we denote the reflexive and transitive closure of the relation $\Longrightarrow_{\mu}$ for $\mu \in \{\mathrm{ex}, \mathrm{in}\}$. The language generated by G is defined as

$$L_{\mu}(G) = \{\, z \mid x \Longrightarrow_{\mu}^{*} z \text{ for some } x \in A \,\}.$$

We omit the index μ if the derivation mode is clear from the context.

By $\mathcal{EC}(\mathcal{F})$, we denote the family of all languages generated externally by contextual grammars with selection in $\mathcal{F}$. When a contextual grammar works in the external mode, we call it an external contextual grammar. By $\mathcal{IC}(\mathcal{F})$, we denote the family of all languages generated internally by contextual grammars with selection in $\mathcal{F}$. When a contextual grammar works in the internal mode, we call it an internal contextual grammar.

3 Results on families of idefix-closed languages

In this section, we investigate inclusion relations between various subregular languages classes. Figure 1 shows the results.

An arrow from a node X to a node Y stands for the proper inclusion $X \subset Y$. If two families are not connected by a directed path, they are incomparable. An edge label refers to the paper where the proper inclusion has been shown (in some cases, it might be that it is not the first paper where the respective inclusion has been mentioned, since it is so obvious that it was not emphasized in a publication) or the lemma of this paper where the proper inclusion will be shown.

In the literature, it is often said that two languages are equivalent if they are equal or differ at most in the empty word. Similarly, two families can be regarded to be equivalent if they differ only in the languages $\emptyset$ or $\{\lambda\}$. Therefore, the set *STAR* of all star languages is sometimes regarded as a proper subset of the set *COM* of all (left-, right-, or two-sided) comet languages although $\{\lambda\}$ belongs to the family *STAR* but not to *LCOM*, *RCOM* or *2COM*. We regard *STAR* and $\mathit{STAR} \setminus \{\{\lambda\}\}$ as different.

We now present some languages which will serve later as witness languages for proper inclusions or incomparabilities.

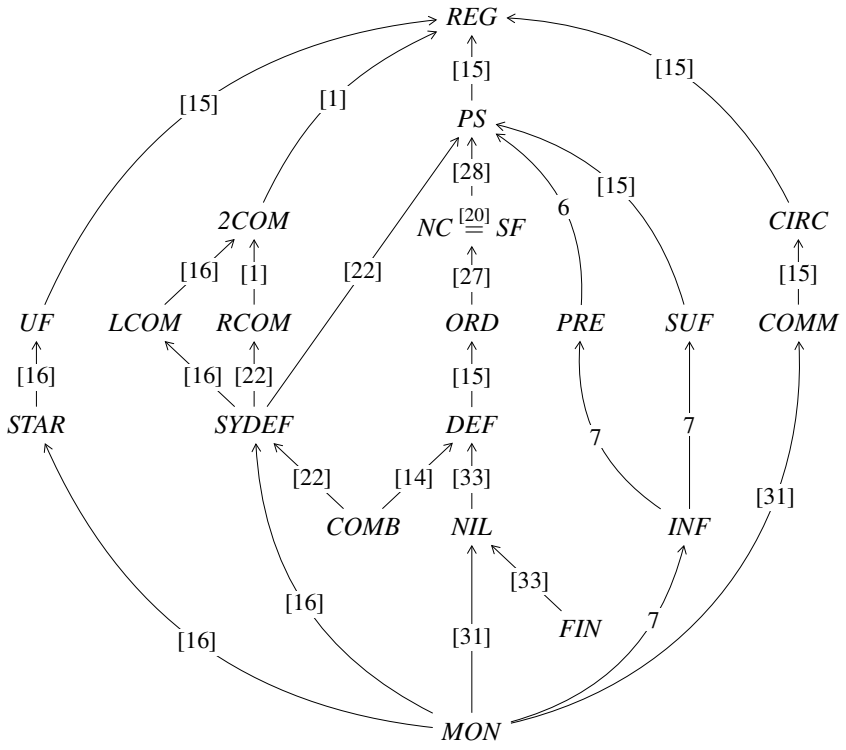

Figure 1: Resulting hierarchy of subregular language families.

Lemma 1 *Let $L = \{ab, a, \lambda\}$. Then, it holds $L \in (FIN \cap PRE) \setminus SUF$.*

Proof. For the word ab, the prefixes a and λ are in the language. For the word a, the prefix λ is in the language. Furthermore, L is finite. Therefore, $L \in FIN \cap PRE$. For the word ab, the suffix b is not in the language. Therefore, $L \notin SUF$ holds. □

Lemma 2 *Let $L = \{ab, b, \lambda\}$. Then, it holds $L \in (FIN \cap SUF) \setminus PRE$.*

Proof. For the word ab, the suffixes b and λ are in the language. For the word b, the suffix λ is in the language. Furthermore, L is finite. Therefore, $L \in SUF$. For the word ab, the prefix a is not in the language. Therefore, $L \notin PRE$ holds. □

Lemma 3 *Let $L = \{ab, a, b, \lambda\}$. Then, it holds $L \in INF \setminus (UF \cup CIRC \cup 2COM)$.*

Proof. For the word ab, all infixes ab, a, b and λ are in the language. For the word b, all infixes b and λ are also in the language. For the word a, all infixes a and λ are in the language, and for the word λ, the infix λ is in the language. It therefore holds that for every word, all infixes are in the language L, so $L \in INF$.

For the word ab, the circular permutation ba is not in the language. Therefore, $L \notin CIRC$. According to [21], a language has an infinite number of words or at most one word if it is union-free. However, this language has four words and therefore $L \notin UF$ holds. Every non-empty language from the family *2COM* is infinite by [16]. Since L is non-empty but finite, we have $L \notin 2COM$. □

Lemma 4 *Let $L = Inf(\{\, ab^{2n}a \mid n \geq 1 \,\})$. Then, it holds $L \in INF \setminus NC$.*

Proof. Since L is the infix-closure of a language, we see $L \in INF$.

Assuming that L is non-counting, it follows from the definition that for all words $x, y, z \in \{a,b\}^*$ and for a number $k \geq 1$, the equivalence $xy^k z \in L \Longleftrightarrow xy^{k+1}z \in L$ applies. We now set $x = z = a$ and $y = b$.

If k is even, $ab^k a \in L$ but $ab^{k+1}a \in L$, which is a contradiction. If k is odd, $ab^{k+1}a \in L$ but $ab^k a \in L$, which is also a contradiction. It follows that $L \notin NC$ holds. □

Lemma 5 *Let $L = \{a,b\}^*\{b\}$. Then, it holds $L \in COMB \setminus PRE$.*

Proof. With $V = \{a,b\}$ and $A = \{b\}$, the language L has the structure $L = V^*A$. Therefore, $L \in COMB$. The word aab is in L but the prefix aa is not. Hence, $L \notin PRE$ holds. □

We now prove some inclusion relations.

Lemma 6 *The proper inclusion $PRE \subset PS$ holds.*

Proof. The inclusion $PRE \subseteq PS$ can be shown similarly to the inclusion $SUF \subset PS$ which was proved in [15]. The language $L = \{a,b\}^*\{b\}$ from Lemma 5 is a witness language since it is in $COMB \setminus PRE$ and therefore in $PS \setminus PRE$. □

Lemma 7 *The proper inclusions $MON \subset INF \subset PRE$ and $INF \subset SUF$ hold.*

Proof. The inclusions $MON \subseteq INF \subseteq PRE$ and $INF \subseteq SUF$ follow from the definition. For their properness, we have the following witness languages:

1. $MON \subset INF$: The language $L = Inf\left(\{\, ab^{2n}a \mid n \geq 1 \,\}\right)$ from Lemma 4 is a witness language since it is in $INF \setminus NC$ and therefore in $INF \setminus MON$ (because $MON \subset NC$).
2. $INF \subset PRE$: The language $L = \{ab, a, \lambda\}$ from Lemma 1 is a witness language since it belongs to the set $PRE \setminus SUF$ and therefore, it also belongs to the set $PRE \setminus INF$.
3. $INF \subset SUF$: The language $L = \{ab, b, \lambda\}$ from Lemma 2 is a witness language since it belongs to the set $SUF \setminus PRE$ and therefore, it also belongs to the set $SUF \setminus INF$. □

We now prove the incomparability relations mentioned in Figure 1 which have not been proved earlier. These are the relations regarding the families *PRE* and *INF*. In most cases, we show the incomparability of whole 'strands' in the hierarchy. For a strand consisting of language families $F_1, F_2, \ldots, F_n$ where F_1 is a subset of every family F_i with $1 \leq i \leq n$ and every such family is a subset of the family F_n and a strand consisting of families $F'_1, F'_2, \ldots, F'_m$ where $F'_1 \subseteq F'_j$ and $F'_j \subseteq F'_m$ for $1 \leq j \leq m$, it suffices to show that there are a language L in $F_1 \setminus F'_m$ and a language L' in $F'_1 \setminus F_n$ in order to show that every family F_i is incomparable to every family F'_j with $1 \leq i \leq n$ and $1 \leq j \leq m$ (because $L \in F_i \setminus F'_j$ and $L' \in F'_j \setminus F_i$). So, we give only two witness languages L and L' for every pair of strands.

Lemma 8 *The language families SUF and PRE are incomparable to each other.*

Proof. Witness languages are given in Lemmas 1 and 2. □

Lemma 9 *Let $\mathcal{F} = \{COMB, DEF, SYDEF, ORD, NC\}$. Every family in $\mathcal{F}$ is incomparable to the families PRE and INF.*

Proof. As witness languages, we have

$$L_1 = Inf\left(\{\, ab^{2n}a \mid n \geq 1 \,\}\right) \in INF \setminus NC \quad \text{and} \quad L_2 = \{a,b\}^*\{b\} \in COMB \setminus PRE$$

from Lemma 4 and Lemma 5, respectively. □

Lemma 10 *Let $\mathcal{F} = \{FIN, NIL\}$. Every family in $\mathcal{F}$ is incomparable to the families PRE and INF.*

Proof. As witness languages, we have

$$L_1 = \{ab, b, \lambda\} \in FIN \setminus PRE \quad \text{and} \quad L_2 = Inf\left(\{\, ab^{2n}a \mid n \geq 1 \,\}\right) \in INF \setminus NC$$

from Lemma 2 and Lemma 4, respectively. □

Lemma 11 *Let $\mathcal{F} = \{SYDEF, RCOM, LCOM, 2COM\}$. Every family in $\mathcal{F}$ is incomparable to the families PRE and INF.*

Proof. As witness languages, we have

$$L_1 = \{ab, a, b, \lambda\} \in INF \setminus 2COM \quad \text{and} \quad L_2 = \{a,b\}^*\{b\} \in COMB \setminus PRE$$

from Lemma 3 and Lemma 5, respectively. □

Lemma 12 *Let $\mathcal{F} = \{STAR, UF\}$. Every family in $\mathcal{F}$ is incomparable to the families PRE and INF.*

Proof. As witness languages, we have

$$L_1 = \{ab, a, b, \lambda\} \in INF \setminus UF \quad \text{and} \quad L_2 = \{aa\}^* \in STAR \setminus PS$$

from Lemma 3 and [16, Lemma 11], respectively. □

Lemma 13 *Let $\mathcal{F} = \{COMM, CIRC\}$. Every family in $\mathcal{F}$ is incomparable to the families PRE and INF.*

Proof. As witness languages, we have

$$L_1 = \{ab, a, b, \lambda\} \in INF \setminus CIRC \quad \text{and} \quad L_2 = \{aa\}^* \in COMM \setminus PS$$

from Lemma 3 and [15, Lemma 4.9]. □

From all these relations, the hierarchy presented in Figure 1 follows. An edge label refers to the paper or lemma in the present paper where the proper inclusion is shown. The incomparability results are proved in Lemmas 8 through 13.

Theorem 14 (Resulting hierarchy for subregular families) *The inclusion relations presented in Figure 1 hold. An arrow from an entry X to an entry Y depicts the proper inclusion $X \subset Y$; if two families are not connected by a directed path, they are incomparable.*

4 Results on subregular control in external contextual grammars

In this section, we include the families of languages generated by external contextual grammars with selection languages from the subregular families under investigation into the existing hierarchy with respect to external contextual grammars.

Lemma 15 (Monotonicity $\mathcal{EC}$) *For any two language classes X and Y with $X \subseteq Y$, we have the inclusion $\mathcal{EC}(X) \subseteq \mathcal{EC}(Y)$.*

Figure 2 shows the inclusion relations between language families which are generated by external contextual grammars where the selection languages belong to subregular classes investigated before. The hierarchy contains results which were already known (marked by a reference to the literature) and results which are new.

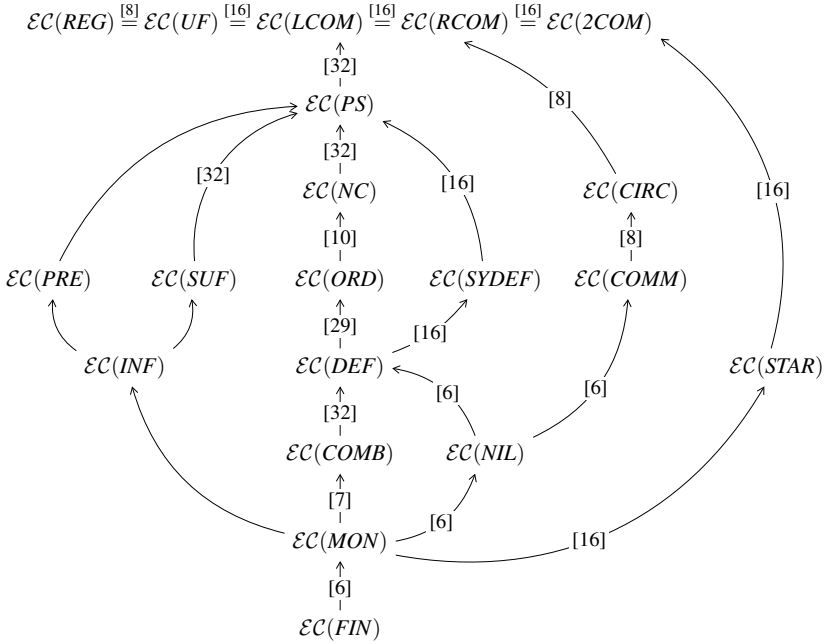

Figure 2: Resulting hierarchy of language families by external contextual grammars with special selection languages.

We now present some languages which serve as witness languages for proper inclusions or incomparabilities.

Lemma 16 *The language $L = \{ a^n b^n \mid n \geq 1 \} \cup \{ b^n \mid n \geq 1 \}$ is in $\mathcal{EC}(PRE) \setminus \mathcal{EC}(SUF)$.*

Proof. The contextual grammar $G = (\{a,b\}, \{S_1 \to C_1, S_2 \to C_2\}, \{ab, b\})$ with

$$S_1 = Pre(\{ a^n b^m \mid n,m \geq 1 \}),\ C_1 = \{(a,b)\},\ S_2 = \{b\}^*,\ C_2 = \{(\lambda, b)\}$$

generates the language L as can be seen as follows.

The first rule can be applied to the axiom ab and then to every other word $a^k b^k$ for $k \geq 1$. This allows us to generate the language $\{ a^n b^n \mid n \geq 1 \}$. From the axiom ab, no other word can be generated. The

second rule can be applied to the axiom b and then to every other word b^k for $k \geq 1$. This allows us to generate the language $\{ b^n \mid n \geq 1 \}$. From the axiom b, no other word can be generated. Together, we obtain that $L(G) = L$. Furthermore, all selection languages are prefix-closed. Hence, $L \in \mathcal{EC}(PRE)$. From [6, Lemma 3.3], we know that the language L is not in $\mathcal{EC}(SUF)$. □

Lemma 17 *The language* $L = \{ a^n b^n \mid n \geq 1 \} \cup \{ a^n \mid n \geq 1 \}$ *is in* $\mathcal{EC}(SUF) \setminus \mathcal{EC}(PRE)$.

Proof. The proof is similar to the one for Lemma 16 due to the symmetry. □

Lemma 18 *The language* $L = \{a,b\}^* \{ a^n b^m \mid n \geq 1, m \geq 1 \} \cup \{ ca^n b^m c \mid n \geq 1, m \geq 1 \}$ *belongs to the family* $\mathcal{EC}(INF) \setminus \mathcal{EC}(STAR)$.

Proof. The contextual grammar $G = (\{a,b\}, \{S_1 \to C_1, S_2 \to C_2\}, \{ab\})$ with

$$S_1 = Inf(\{ a^n b^m \mid n \geq 1, m \geq 1 \}),\ C_1 = \{(c,c)\},\ S_2 = Inf(\{a,b\}^*),\ C_2 = \{(a,\lambda),(b,\lambda),(\lambda,b)\}$$

generates the language L where all selection languages are infix-closed. With star languages as selection languages, the structure of a word cannot be checked before adjoining the letter c. □

Lemma 19 *Let* $L = \{ a^m b c^{2n} b a^m \mid n \geq 1, m \geq 0 \} \cup \{ c^n \mid n \geq 2 \} \cup \{ bc^n b \mid n \geq 2 \} \cup \{ ac^n a \mid n \geq 2 \}$. *Then,* $L \in \mathcal{EC}(INF) \setminus \mathcal{EC}(NC)$.

Proof. The contextual grammar $G = (\{a,b,c\}, \{S_1 \to C_1, S_2 \to C_2\}, \{cc\})$ with

$$S_1 = \{c\}^*,\ C_1 = \{(\lambda,c),(b,b)\},\ S_2 = Inf(\{ a^m b c^{2n} b a^m \mid n \geq 1, m \geq 0 \}),\ C_2 = \{(a,a)\}$$

generates the language L and all selection languages are infix-closed. With non-counting selection languages, one could also wrap letters a around a word $bc^n b \in L$ for an odd number n which is a contradiction. □

Lemma 20 *The language* $L = \{a,b\}^* \cup \{c\}\{\lambda,b\}\{ab\}^*\{\lambda,a\}\{c\}$ *is in* $\mathcal{EC}(INF) \setminus \mathcal{EC}(SYDEF)$.

Proof. The language L is generated by the contextual grammar $G = (\{a,b,c\}, \{S_1 \to C_1, S_2 \to C_2\}, \{\lambda\})$ with

$$S_1 = \{a,b\}^*,\ C_1 = \{(\lambda,a),(\lambda,b)\},\ S_2 = Inf(\{ab\}^*),\ C_2 = \{(c,c)\}$$

All selection languages are infix-closed; therefore, we have $L \in \mathcal{EC}(INF)$. With symmetric definite selection languages, the alternation between a and b cannot be checked. □

Lemma 21 *The language* $L = \{ a^n b^n \mid n \geq 1 \} \cup \{ b^n a^n \mid n \geq 1 \}$ *is in* $\mathcal{EC}(INF) \setminus \mathcal{EC}(CIRC)$.

Proof. The contextual grammar $G = (\{a,b\}, \{S_1 \to C_1, S_2 \to C_2\}, \{ab,ba\})$ with

$$S_1 = Inf(\{a\}^*\{b\}^*),\ C_1 = \{(a,b)\},\ S_2 = Inf(\{b\}^*\{a\}^*),\ C_2 = \{(b,a)\}$$

generates L where all selection languages are infix-closed. With circular selection languages, a word of the language $\{a\}^+\{b\}^+\{a\}^+\{b\}^+$ could be generated. □

Lemma 22 *The language* $L = \{ b^n a \mid n \geq 0 \} \cup \{\lambda\}$ *is in* $\mathcal{EC}(COMB) \setminus \mathcal{EC}(PRE)$.

Proof. The contextual grammar $G = (\{a,b\}, \{\{a,b\}^*\{a\} \to (b,\lambda)\}, \{\lambda, a\})$ generates the language L and all the selection languages are combinational. With prefix-closed selection languages, also words without a could be generated. □

Lemma 23 *The language* $L = \{ bbba^n \mid n \geq 1 \} \cup \{bb\}$ *is in* $\mathcal{EC}(NIL) \setminus \mathcal{EC}(PRE)$.

Proof. The contextual grammar $G = (\{a,b\}, \{\{a,b\}^4\{a,b\}^* \to (\lambda,a)\}, \{bbba, bb\})$ generates the language L, where all selection languages in $\mathcal{S}$ are nilpotent. With prefix-closed selection languages, also a word bba^m could be generated. □

With the languages from the previous lemmas, the inclusion relations and incomparabilities depicted in Figure 2 can be shown.

Theorem 24 (Resulting hierarchy for $\mathcal{EC}$) *The inclusion relations presented in Figure 2 hold. An arrow from an entry X to an entry Y depicts the proper inclusion* $X \subset Y$*; if two families are not connected by a directed path, they are incomparable.*

5 Results on subregular control in internal contextual grammars

In this section, we include the families of languages generated by internal contextual grammars with selection languages from the subregular families under investigation into the existing hierarchy with respect to internal contextual grammars.

Lemma 25 (Monotonicity $\mathcal{IC}$) *For any two language classes X and Y with* $X \subseteq Y$*, we have the inclusion* $\mathcal{IC}(X) \subseteq \mathcal{IC}(Y)$.

Figure 3 shows a hierarchy of some language families which are generated by internal contextual grammars where the selection languages belong to subregular classes investigated before. The hierarchy contains results which were already known (marked by a reference to the literature) and results which are new.

We now present some languages which serve as witness languages for proper inclusions or incomparabilities.

Lemma 26 *Let* $G = (\{a,b,c,d\}, \{\{ab,b,\lambda\} \to (c,d)\}, \{aab\})$ *be a contextual grammar. Then, the language* $L = L(G)$ *is in* $\mathcal{IC}(SUF) \setminus \mathcal{IC}(PRE)$.

Proof. Since the selection language of G is suffix-closed, the language L is in $\mathcal{IC}(SUF)$. Suppose that the language L is also generated by a contextual grammar $G' = (\{a,b,c,d\}, \mathcal{S}', B')$ where all selection languages $S \in \mathcal{S}'$ are prefix-closed.

Let us consider a word $w = ac^n abd^n \in L$ for some natural number $n \geq \ell(G')$. Due to the choice of n, the word w is derived in one step from some word $z_1z_2z_3 \in L$ for three words $z_i \in V^*$ with $1 \leq i \leq 3$ by using a selection component $(S,C) \in \mathcal{S}$ with $z_2 \in S$ and a context $(u,v) \in C$: $z_1z_2z_3 \Longrightarrow z_1uz_2vz_3 = w$. By the structure of the language L, it holds $(u,v) = (c^m, d^m)$ for a natural number m with $1 \leq m < n$ and $z_2 = c^p abd^q$ for two natural numbers p and q with $m+p \leq n$ and $q+m \leq n$. Since S is assumed

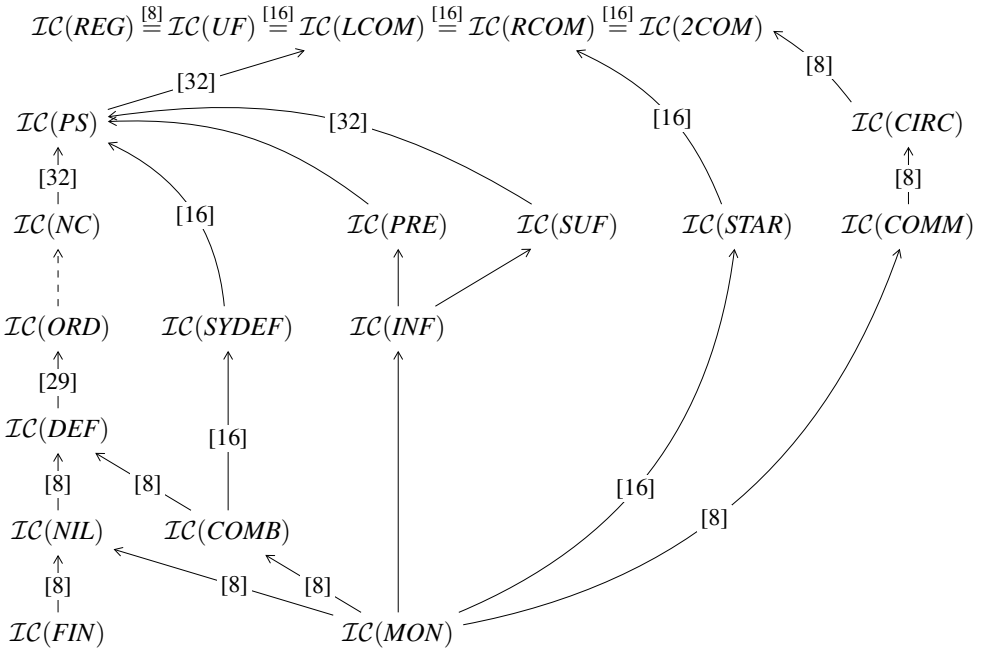

Figure 3: Resulting hierarchy of language families by internal contextual grammars with special selection languages

to be prefix-closed, the word $c^p a$ is in S, too. Therefore, we can apply the context (u,v) to the subword $c^p a$ of w. Hence, we can derive $ac^n abd^n \in L$ to $ac^{n+m} ad^m bd^n \notin L$. This contradiction proves that $L \notin \mathcal{IC}(PRE)$. □

Lemma 27 *Let $G = (\{a,b,c,d\}, \{\{ab,a,\lambda\} \to (c,d)\}, \{abb\})$ be a contextual grammar. Then, the language $L = L(G)$ is in $\mathcal{IC}(PRE) \setminus \mathcal{IC}(SUF)$.*

Proof. The argumentation is symmetrical to the previous proof. □

Lemma 28 *Let $V = \{a,b,c,d,e,f,g,h\}$ be an alphabet, $G = (V, \{S_1 \to C_1, S_2 \to C_2\}, \{cd\})$ be a contextual grammar with*

$$S_1 = Inf(\{a,b\}^*\{cd\}),\ C_1 = \{(aab,gh)\},\ S_2 = Inf(\{a\}\{bb\}^+\{c\}),\ C_2 = \{(e,f)\},$$

and $L = L(G)$ be its generated language. Then, $L \in \mathcal{IC}(INF) \setminus \mathcal{IC}(NC)$.

Proof. All selection languages are infix-closed, therefore, $L \in \mathcal{IC}(INF)$. With non-counting selection languages, it could not be ensured that the number of letters b between e and f in a word is even. □

Lemma 29 *Let $V = \{a,b,c\}$ be an alphabet, $G = (V, \{Inf(\{abca\}) \to (b,c)\}, \{abcaaabca\})$ be a contextual grammar, and $L = L(G)$ be its generated language. Then, $L \in \mathcal{IC}(INF) \setminus \mathcal{IC}(SYDEF)$.*

Proof. The selection language of the given contextual grammar is infix-closed; therefore, $L \in \mathcal{IC}(INF)$. With symmetric definite selection languages, letters b could be produced in the beginning of a word whereas the corresponding letters c are produced at the very end of a word which is a contradiction. □

Lemma 30 *Let $V = \{a,b,c,d\}$ be an alphabet, $G = (V, \{\{ab, a, b, \lambda\} \to (c,d)\}, \{aab, ba\})$ be a contextual grammar, and $L = L(G)$ be the language generated. Then, $L \in \mathcal{IC}(INF) \setminus \mathcal{IC}(CIRC)$.*

Proof. The selection language of G is infix-closed. Hence, we have $L \in \mathcal{IC}(INF)$. In [9, Lemma 17], it was proved that the language L is not in $\mathcal{IC}(CIRC)$. □

Lemma 31 *Let $L = \{\, c^n a c^m b c^{n+m} \mid n \geq 0, m \geq 0 \,\}$. Then, $L \in (\mathcal{IC}(FIN) \cap \mathcal{IC}(COMB)) \setminus \mathcal{IC}(PRE)$.*

Proof. The relations $L \in \mathcal{IC}(FIN)$ and $L \in \mathcal{IC}(COMB)$ were proved in [9, Lemma 15]. The relation $L \notin \mathcal{IC}(PRE)$ can be proved in the same way as $L \notin \mathcal{IC}(SUF)$ is proved in [9, Lemma 15]. □

Lemma 32 *Let $V = \{a,b,c,d\}$ be an alphabet, $G = (V, \{S_1 \to C_1, S_2 \to C_2\}, \{baab\})$ be a contextual grammar with $S_1 = \{a, \lambda\}$, $C_1 = \{(c,d)\}$, $S_2 = \{b, \lambda\}$, $C_2 = \{(d,c)\}$, and $L = L(G)$ its generated language. Then, $L \in \mathcal{IC}(INF) \setminus \mathcal{IC}(STAR)$.*

Proof. Since both selection languages of G are infix-closed, we have $L \in \mathcal{IC}(INF)$. In [17], it is proved that $L \notin \mathcal{IC}(STAR)$. □

With the languages from the previous lemmas, the inclusion relations and incomparabilities depicted in Figure 3 can be shown.

Theorem 33 (Resulting hierarchy for $\mathcal{IC}$) *The inclusion relations presented in Figure 3 hold. An arrow from an entry X to an entry Y depicts the proper inclusion $X \subset Y$; if two families are not connected by a directed path, they are incomparable.*

Please note that with the result in Theorem 33, we have answered the open question whether $\mathcal{IC}(SUF)$ is incomparable to $\mathcal{IC}(NC)$ or $\mathcal{IC}(ORD)$ or whether it is a subset of one of the families $\mathcal{IC}(NC)$ or $\mathcal{IC}(ORD)$ raised already several years ago in [30].

6 Conclusion and future work

In this paper, we have extended the previous hierarchies of subregular language families, of families generated by external contextual grammars with selection in certain subregular language families, and of families generated by internal contextual grammars with selection in such language families.

Various other subregular language families have also been investigated in the past (for instance, in [1, 13, 22]). Future research will be on extending and unifying current hierarchies of subregular language families (presented, for instance, in [10, 32]) by additional families and to use them as control in contextual grammars.

The extension of the hierarchy with other families of definite-like languages (for instance, ultimate definite, central definite, non-inital definite) has also already begun. Furthermore, it is also planned to unify the hierarchy of subregular language families, extended by the mentioned language families, with the hierarchies of the language families generated by contextual grammars defined by their limited resources.

The research can be also extended to other mechanisms like tree-controlled grammars or networks of evolutionary processors. Another possibility would be to check to what extent the different language classes are closed under different operations.

References

[1] Henning Bordihn, Markus Holzer & Martin Kutrib (2009): *Determination of finite automata accepting subregular languages*. *Theoretical Computer Science* 410(35), pp. 3209–3222, doi:10.1016/j.tcs.2009.05.019.

[2] Janusz A. Brzozowski (1962): *Regular expression techniques for sequential circuits*. Ph.D. thesis, Princeton University, Princeton, NJ, USA.

[3] Janusz A. Brzozowski (1967): *Roots of star events*. *Journal of the ACM* 14(3), pp. 466–477, doi:10.1109/SWAT.1966.21.

[4] Janusz A. Brzozowski & Rina Cohen (1969): *On decompositions of regular events*. *Journal of the ACM* 16(1), pp. 132–144, doi:10.1145/321495.321505.

[5] Janusz A. Brzozowski, Galina Jirásková & Chenglong Zou (2014): *Quotient complexity of closed languages*. *Theory of Computing Systems* 54, pp. 277–292, doi:10.1007/s00224-013-9515-7.

[6] Jürgen Dassow (2005): *Contextual grammars with subregular choice*. *Fundamenta Informaticae* 64(1–4), pp. 109–118.

[7] Jürgen Dassow (2015): *Contextual languages with strictly locally testable and star free selection languages*. *Analele Universitatii Bucuresti* 62, pp. 25–36.

[8] Jürgen Dassow, Florin Manea & Bianca Truthe (2012): *On external contextual grammars with subregular selection languages*. *Theoretical Computer Science* 449, pp. 64–73, doi:10.1016/j.tcs.2012.04.008.

[9] Jürgen Dassow, Florin Manea & Bianca Truthe (2012): *On Subregular Selection Languages in Internal Contextual Grammars*. *Journal of Automata, Languages, and Combinatorics* 17(2–4), pp. 145–164, doi:10.25596/jalc-2012-145.

[10] Jürgen Dassow & Bianca Truthe (2023): *Relations of contextual grammars with strictly locally testable selection languages*. *RAIRO – Theoretical Informatics and Applications* 57, p. #10, doi:10.1051/ita/2023012.

[11] Ference Gécseg & István Peák (1972): *Algebraic Theory of Automata*. Academiai Kiado, Budapest.

[12] Arthur Gill & Lawrence T. Kou (1974): *Multiple-entry finite automata*. *Journal of Computer and System Sciences* 9(1), pp. 1–19, doi:10.1016/S0022-0000(74)80034-6.

[13] Yo-Sub Han & Kai Salomaa (2009): *State complexity of basic operations on suffix-free regular languages*. *Theoretical Computer Science* 410(27), pp. 2537–2548, doi:10.1016/j.tcs.2008.12.054.

[14] Ivan M. Havel (1969): *The theory of regular events II*. *Kybernetika* 5(6), pp. 520–544.

[15] Markus Holzer & Bianca Truthe (2015): *On relations between some subregular language families*. In Rudolf Freund, Markus Holzer, Nelma Moreira & Rogério Reis, editors: *Seventh Workshop on Non-Classical Models of Automata and Applications – NCMA 2015, Porto, Portugal, August 31 – September 1, 2015. Proceedings*, *books@ocg.at* 318, Österreichische Computer Gesellschaft, pp. 109–124.

[16] Marvin Ködding & Bianca Truthe (2024): *Various Types of Comet Languages and their Application in External Contextual Grammars*. In Florin Manea & Giovanni Pighizzini, editors: *Proceedings 14th International Workshop on Non-Classical Models of Automata and Applications (NCMA 2024), NCMA 2024, Göttingen, Germany, 12–13 August 2024*, *EPTCS* 407, pp. 118–135, doi:10.4204/EPTCS.407.9.

[17] Marvin Ködding & Bianca Truthe (submitted): *Various Types of Comet Languages and Their Application in Contextual Grammars*. *Journal of Automata, Languages, and Combinatorics*.

[18] Manfred Kudlek (2004): *On languages of cyclic words*. In Natasha Jonoska, Gheorghe Păun & Grzegorz Rozenberg, editors: *Aspects of Molecular Computing, Essays Dedicated to Tom Head on the Occasion of His 70th Birthday*, *LNCS* 2950, Springer-Verlag, pp. 278–288, doi:10.1007/978-3-540-24635-0_20.

[19] Solomon Marcus (1969): *Contextual grammars*. *Revue Roumaine de Mathématique Pures et Appliquées* 14, pp. 1525–1534.

[20] Robert McNaughton & Seymour Papert (1971): *Counter-Free Automata*. MIT Press, Cambridge, USA.

[21] Benedek Nagy (2019): *Union-freeness, deterministic union-freeness and union-complexity*. In Michal Hospodár, Galina Jirásková & Stavros Konstantinidis, editors: *Descriptional Complexity of Formal Systems, 21st IFIP WG 1.02 International Conference, DCFS 2019, Košice, Slovakia, July 17–19, 2019, Proceedings*, Springer, Cham, pp. 46–56, doi:10.1007/978-3-030-23247-4_3.

[22] Viktor Olejár & Alexander Szabari (2023): *Closure Properties of Subregular Languages Under Operations*. *International Journal of Foundations of Computer Science*, pp. 1–25, doi:10.1142/S0129054123450016.

[23] Azaria Paz & Bezalel Peleg (1965): *Ultimate-definite and symmetric-definite events and automata*. *Journal of the ACM* 12(3), pp. 399–410, doi:10.1145/321281.321292.

[24] Micha A. Perles, Michael O. Rabin & Eli Shamir (1963): *The theory of definite automata*. *IEEE Transactions of Electronic Computers* 12, pp. 233–243, doi:10.1109/PGEC.1963.263534.

[25] Grzegorz Rozenberg & Arto Salomaa, editors (1997): *Handbook of Formal Languages*. Springer-Verlag, Berlin, doi:10.1007/978-3-642-59136-5.

[26] Huei-Jan Shyr (1991): *Free Monoids and Languages*. Hon Min Book Co., Taichung, Taiwan.

[27] Huei-Jan Shyr & Gabriel Thierrin (1974): *Ordered automata and associated languages*. *Tamkang Journal of Mathematics* 5(1), pp. 9–20.

[28] Huei-Jan Shyr & Gabriel Thierrin (1974): *Power-separating regular languages*. *Mathematical Systems Theory* 8(1), pp. 90–95, doi:10.1007/BF01761710.

[29] Bianca Truthe (2014): *A relation between definite and ordered finite automata*. In Suna Bensch, Rudolf Freund & Friedrich Otto, editors: *Sixth Workshop on Non-Classical Models for Automata and Applications – NCMA 2014, Kassel, Germany, July 28–29, 2014. Proceedings*, *books@ocg.at* 304, Österreichische Computer Gesellschaft, pp. 235–247.

[30] Bianca Truthe (2017): *Hierarchies of Language Families of Contextual Grammars*. In Rudolf Freund, František Mráz & Daniel Průša, editors: *Nineth Workshop on Non-Classical Models of Automata and Applications (NCMA), Prague, Czech Republic, August 17–18, 2017, Proceedings*, *books@ocg.at* 329, Österreichische Computer Gesellschaft, pp. 13–28.

[31] Bianca Truthe (2018): *Hierarchy of Subregular Language Families*. Technical Report, Justus-Liebig-Universität Giessen, Institut für Informatik, IFIG Research Report 1801.

[32] Bianca Truthe (2021): *Generative capacity of contextual grammars with subregular selection languages*. *Fundamenta Informaticae* 180(1–2), pp. 123–150, doi:10.3233/FI-2021-2037.

[33] Barbara Wiedemann (1978): *Vergleich der Leistungsfähigkeit endlicher determinierter Automaten*. Diplomarbeit, Universität Rostock.

On a Generalization of the Christoffel Tree: Epichristoffel Trees

Abhishek Krishnamoorthy
Madras Christian College
Chennai, Tamil Nadu, India
Department of Mathematics
abhishek@mcc.edu.in

Robinson Thamburaj
Madras Christian College
Chennai, Tamil Nadu, India
Department of Mathematics
robinson@mcc.edu.in

Durairaj Gnanaraj Thomas
Madras Christian College
Chennai, Tamil Nadu, India
Department of Mathematics
dgthomasmcc@yahoo.com

Sturmian words form a family of one-sided infinite words over a binary alphabet that are obtained as a discretization of a line with an irrational slope starting from the origin. A finite version of this class of words called Christoffel words has been extensively studied for their interesting properties. It is a class of words that has a geometric and an algebraic definition, making it an intriguing topic of study for many mathematicians. Recently, a generalization of Christoffel words for an alphabet with 3 letters or more, called epichristoffel words, using episturmian morphisms has been studied, and many of the properties of Christoffel words have been shown to carry over to epichristoffel words; however, many properties are not shared by them as well. In this paper, we introduce the notion of an epichristoffel tree, which proves to be a useful tool in determining a subclass of epichristoffel words that share an important property of Christoffel words, which is the ability to factorize an epichristoffel word as a product of smaller epichristoffel words. We also use the epichristoffel tree to present some interesting results that help to better understand epichristoffel words.

1 Introduction

Sturmian sequences have appeared several times in history in the works of several mathematicians such as Bernoulli, Christoffel and Markov. They are sequences over a 2-letters alphabet that code discrete lines, due to which they appear in different fields of mathematics such as discrete geometry and number theory. They also have the property of being balanced. Christoffel words are the finite version of these sequences and have been studied extensively [1, 3, 6, 9, 10, 11]. For a comprehensive understanding of Christoffel words, we refer the readers to [9],[13] and [14]. These are the smallest words with respect to the lexicographic order which are conjugate to a finite standard Sturmian word.

Genevieve Paquin has studied a generalization of Christoffel words called epichristoffel words, in which episturmian morphisms are used to determine if a word belongs to an epichristoffel class [7]. Although epichristoffel words share many of the same properties as Christoffel words, Genevieve raised some open problems regarding epichristoffel words. These include the ability to characterize the epichristoffel word of each conjugacy class, whether epichristoffel words satisfy a type of balanced property, and whether there is an epichristoffel word of any length over a k-letter alphabet for a fixed $k \geq 3$. In this paper, we have partially answered such questions by introducing infinite binary trees called epichristoffel trees motivated by the definition of the Christoffel tree. We note that there can be different epichristoffel trees based on the epichristoffel word that serves as the root of the tree. It has been shown in [2] that every non-trivial Christoffel word can be factorized in a unique way into two words where each one of them is a Christoffel word. This is called the standard factorization of the word. The Christoffel tree is an infinite binary tree in which each Christoffel word appears exactly once in its standard factorization form. Although the epichristoffel trees that we introduce in this article do not contain

N. Moreira and L. Prigioniero (Eds.): 15th International Workshop on Non-Classical Models of Automata and Applications (NCMA 2025).
EPTCS 422, 2025, pp. 15–28, doi:10.4204/EPTCS.422.2

every epichristoffel word over a k-letter alphabet and not all epichristoffel words have a factorization into smaller epichristoffel words, we show that the epichristoffel tree can be a useful tool in helping to determine the existence of epichristoffel words of various lengths. This tree exhibits subclasses of epichristoffel words that do or do not satisfy a factorization property, such as Christoffel words. Through Theorem 5.1, we have shown that this tree provides a characterization of the epichristoffel word of a conjugacy class. The study of epichristoffel words is interesting because they seem to relate to the Fraenkel conjecture which states that for a k-letter alphabet, there exists a unique infinite word up to letter permutation and conjugation that is balanced and has pair-wise distinct letter frequencies. Thus, a better understanding of epichristoffel words might help prove this conjecture.

2 Definitions and Notations

Throughout this paper, we use the notation A to denote a finite ordered alphabet. A finite word is an element of the free monoid A^* and if $w = w[0]w[1]\dots w[n-1]$, with $w[i] \in A$ then w is said to be a finite word of length n. Unless specified otherwise every word discussed in this paper will be a finite word. The notation $|w|$ is used to denote the length of the word w and the notation $|w|_{a_i}$ is used to determine the number of occurrences of the letter a_i in w, where $a_i \in A$. By convention, the empty word is denoted by λ and its length is 0.

The conjugacy class $[w]$ of a finite word w of length n is the set of all words of the form $w[i]w[i+1]\dots w[n-1]w[0]\dots w[i-1]$, for $0 \leq i \leq n-1$. If two words w_1 and w_2 are conjugate to each other, we denote it by $w_1 \equiv w_2$. If w is primitive and the smallest word in its conjugacy class with respect to the lexicographic order, then w is called a Lyndon word.

A word f is a factor of a word w if $w = pfs$ for some $p,s \in A^*$. f is called a prefix or a suffix respectively if p or s is the empty word. We refer the readers to [7] and [9] for most of the basic notations and definitions used in this paper.

3 Christoffel Words and Epichristoffel Words

Definition 3.1. [9] The lower Christoffel path of slope $\frac{a}{b}$ over the binary alphabet $A = \{x,y\}$, where a and b are relatively prime positive integers is the path in the plane from (0, 0) to (b,a) in the integer lattice $Z \times Z$ that satisfies the following two conditions:

- The path lies below the line segment that begins at (0, 0) and ends at (b,a).
- The region in the plane enclosed by the path and the line segment contains no other part of $Z \times Z$ besides those of the path.

By encoding every horizontal step in the lower Christoffel path by the letter x and every vertical step in the lower Christoffel path by the letter y we get word of length $a+b$ over a binary alphabet called the Christoffel word of slope $\frac{a}{b}$.

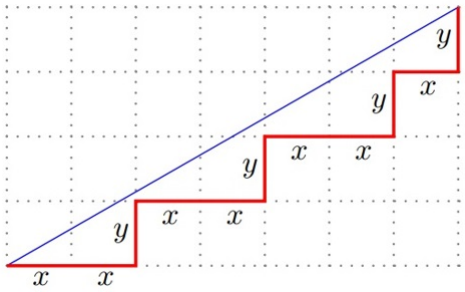

Figure 1: The Christoffel word of slope $\frac{4}{7}$

Lemma 3.1. *[6] A word w is a Christoffel word if and only if w is a balanced Lyndon word.*

Definition 3.2. [9] The label of a point (i, j) on the lower Christoffel path of slope $\frac{a}{b}$ is the number $\frac{ia-jb}{b}$ which represents the vertical distance from the point (i, j) to the line segment from (0, 0) to (b, a).

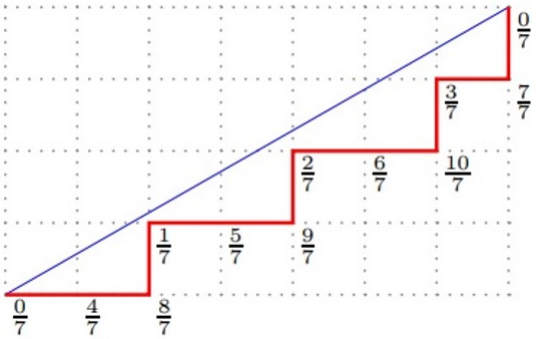

Figure 2: The labels of the points on the Christoffel path of slope $\frac{4}{7}$

Definition 3.3. [9] The standard factorization of the Christoffel word w of slope $\frac{a}{b}$ is the factorization $w = (w_1, w_2)$ where w_1 encodes the portion of the Christoffel path from (0, 0) to the closest point C on the path having label $\frac{1}{b}$ and w_2 encodes the portion from C to (b, a).

Definition 3.4. [8] For a finite alphabet A and $a_1, a_2 \in A$, consider the following endomorphisms of A^*:

(i) $\psi_{a_1}(a_1) = \overline{\psi}_{a_1}(a_1) = a_1$

(ii) $\psi_{a_1}(a_k) = a_1 a_k$ if $a_k \in A \setminus \{a_1\}$

(iii) $\overline{\psi}_{a_1}(a_k) = a_k a_1$ if $a_k \in A \setminus \{a_1\}$

(iv) $\theta_{a_1 a_2}(a_1) = a_2$, $\theta_{a_1 a_2}(a_2) = a_1$, $\theta_{a_1 a_2}(a_k) = a_k$, $a_k \in A \setminus \{a_1, a_2\}$

Definition 3.5. [7] The set of episturmian morphisms is the monoid generated by the morphisms $\psi_{a_1}, \overline{\psi}_{a_1}$, $\theta_{a_1 a_2}$ under composition. The set of pure episturmian morphisms is the submonoid generated by ψ_{a_1} and $\overline{\psi}_{a_1}$.

Definition 3.6. [7] A word $w \in A^*$ belongs to an epichristoffel class if it is the image of a letter by an episturmian morphism.

Definition 3.7. [7] A word $w \in A^*$ is epichristoffel if it is the unique Lyndon word occurring in an epichristoffel class. A word is called c-epichristoffel if it is conjugate to an epichristoffel word.

Definition 3.8. [7] If w is an epichristoffel word over the alphabet $A = \{a_0, a_1, ..., a_{k-1}\}$ then the k-tuple $p = (p_0, p_1, \ldots, p_{k-1})$ is called an epichristoffel k-tuple if $p_i = |w|_{a_i}$ for $0 \leq i \leq k-1$.

It has been shown in [9] that for a given pair of positive integers a, b there exists a Christoffel word having a number of x's and b number of y's if and only if a and b are relatively prime. For a k-letter alphabet, $k \geq 3$, Paquin presented an algorithm to determine the existence of an epichristoffel k-tuple based on the following definition and results.

Definition 3.9. [7] Let $p = (p_0, p_1, \ldots, p_{k-1})$ be a k-tuple of non-negative integers. The operator $T : N^k \to Z^k$ is defined over the k-tuple p as

$T(p) = T(p_0, p_1, \ldots, p_{k-1}) = (p_0, p_1, \ldots, p_{i-1}, (p_i - \sum_{j=0, j\neq i}^{k-1} p_j), p_{i+1}, \ldots, p_{k-1})$, where $p_i \geq p_j, \forall\ j \neq i$.

Proposition 3.1. *[7] Let p be a k-tuple. There exists an epichristoffel word with occurrence number of letters p if and only if iterating T over p yields a k-tuple p' with $p'_j = 0$ for $j \neq m$ and $p'_m = 1$, for a unique m such that $0 \leq m \leq k-1$.*

Example 3.1. *Consider the 3-tuple (2, 3, 7). Then, $T(2,3,7) = (2,3,2)$, $T^2(2,3,7) = T(2,3,2) = (2,-1,2)$. Hence there exists no epichristoffel word corresponding to the 3-tuple (2, 3, 7).*

On the other hand, there exists an epichristoffel word corresponding to the 3-tuple (1, 4, 2) since $T(1,4,2) = (1,1,2)$, $T^2(1,4,2) = T(1,1,2) = (1,1,0)$, $T^3(1,4,2) = T(1,1,0) = (1,0,0)$.

Lemma 3.2. *[7] Let $w \in A^*$ be a c-epichristoffel word. Then, there exists a c-epichristoffel word $u \in A^*$, $|u| > 1$ and an episturmian morphism $\phi \in \{\psi_{a_0}, \overline{\psi}_{a_0}\}$, with $a_0 \in A$, such that $w = \phi(u)$ if and only if $|w|_{a_0} > |w|_{a_i}$ for all $a_i \in A$, $i \neq 0$.*

We refer the readers to the iteration found in the proof of Lemma 3.2, as the algorithm used to determine the epichristoffel word is based on this iteration. Given below is an example of constructing the epichristoffel word based on the mentioned iteration.

Example 3.2. *If $A = \{x, y, z\}$, then, for the 3-tuple (1, 4, 2) describing the occurrence numbers of x, y and z respectively, the construction of the epichristoffel word is as follows:*
$(1,4,2) \to^y (1,1,2) \to^z (1,1,0) \to^y (1,0,0)$
$\psi_y\psi_z\psi_y(x) = \psi_y\psi_z(yx) = \psi_y(zyzx) = yzyyzyx$
Since this is obtained by a standard episturmian morphism to a letter, this standard episturmian word is a representative of the epichristoffel conjugacy class. The word yzyyzyx is thus a c-epichristoffel word and the smallest word with respect to the lexicographic order in its conjugacy class that is, xyzyyzy is the epichristoffel word, assuming $x < y < z$.

Lemma 3.3. *[12] A Christoffel word can always be written as the product of two Christoffel words.*

Lemma 3.4. *[7] An epichristoffel word cannot always be written as the product of two epichristoffel words.*

Example 3.3. *Consider the epichristoffel word xzyzzyz over the alphabet $A = \{x, y, z\}$. There are no factorizations of this word into epichristoffel words.*

Lemma 3.5. *[7] Any c-epichristoffel word w of length $n > 1$, can be non-uniquely written as the product of two c-epichristoffel words.*

Example 3.4. *A factorization of the c-epichristoffel word yzyyzyx is yzy, yzyx obtained by considering the iteration in Example 3.2 in the following way: $\psi_y\psi_z\psi_y(x) = \psi_y\psi_z(yx) = \psi_y\psi_z(y), \psi_y\psi_z(x)$*
$= \psi_y(zy), \psi_y(zx) = yzy, yzyx$.
This factorization is used in the later part of the paper to construct an epichristoffel tree.

4 The Christoffel and Stern-Brocot trees

Definition 4.1. [9] The Christoffel tree is an infinite binary tree where each node is of the form (u,v) which represents a Christoffel word occurring in its standard factorization form and the left, right descendants of which are (u,uv) and (uv,v) respectively. The root of this tree is the Christoffel word of slope $\frac{1}{1}$, i.e. (x,y).

Lemma 4.1. *[9] Every Christoffel word appears exactly once on the Christoffel tree.*

Definition 4.2. [9] The mediant of two fractions $\frac{a}{b}$ and $\frac{c}{d}$, denoted by $\frac{a}{b} \oplus \frac{c}{d}$ is $\frac{a+c}{b+d}$. This operation gives rise to the Stern-Brocot sequence as follows:
Let S_0 denote the sequence $\frac{0}{1}, \frac{1}{0}$ (we view $\frac{1}{0}$ as a formal fraction for the purpose of the construction of the successive terms of the sequence). For $i > 0$, S_i is constructed from S_{i-1} by inserting between consecutive elements of the sequence their mediant. The first few iterations of this process yields the following sequence:
$\frac{0}{1}, \mathbf{\frac{1}{1}}, \frac{1}{0} \quad (S_1)$
$\frac{0}{1}, \mathbf{\frac{1}{2}}, \frac{1}{1}, \mathbf{\frac{2}{1}}, \frac{1}{0} \quad (S_2)$
$\frac{0}{1}, \mathbf{\frac{1}{3}}, \frac{1}{2}, \mathbf{\frac{2}{3}}, \frac{1}{1}, \mathbf{\frac{3}{2}}, \frac{2}{1}, \mathbf{\frac{3}{1}}, \frac{1}{0} \quad (S_3)$
$\frac{0}{1}, \mathbf{\frac{1}{4}}, \frac{1}{3}, \mathbf{\frac{2}{5}}, \frac{1}{2}, \mathbf{\frac{3}{5}}, \frac{2}{3}, \mathbf{\frac{3}{4}}, \frac{1}{1}, \mathbf{\frac{4}{3}}, \frac{3}{2}, \mathbf{\frac{5}{3}}, \frac{2}{1}, \mathbf{\frac{5}{2}}, \frac{3}{1}, \mathbf{\frac{4}{1}}, \frac{1}{0} \quad (S_4)$
We have indicated the mediants obtained in each iteration in bold.

Definition 4.3. [9] The Stern-Brocot tree is an infinite binary tree in which the vertices of the i^{th} $(i > 0)$ level are the mediants obtained in the i^{th} iteration of the Stern-Brocot sequence.

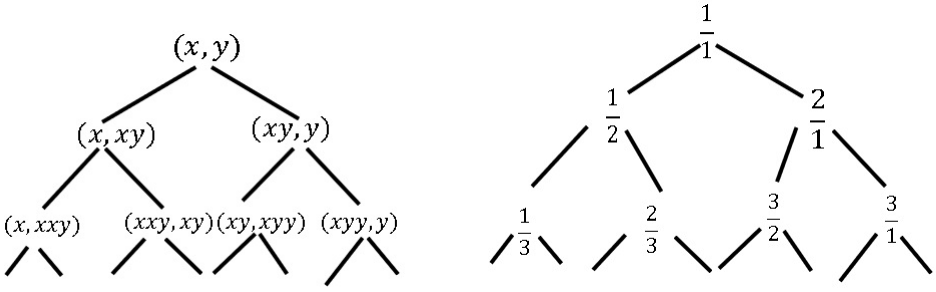

Figure 3: The Christoffel and Stern-Brocot trees

Theorem 4.1. *[9] The Christoffel tree is isomorphic to the Stern-Brocot tree via the map that associates to the vertex (u,v) of the Christoffel tree, the fraction $\frac{|uv|_y}{|uv|_x}$. The inverse map associates to a fraction $\frac{a}{b}$ the pair (u,v) where (u,v) is the standard factorization of the Christoffel word of slope $\frac{a}{b}$.*

Definition 4.4. [5] For a natural number k, the k^{th} left diagonal of the Stern-Brocot tree L_k is the sequence made up of each k^{th} term from each level beginning at the first level and the k^{th} right diagonal of the Stern-Brocot tree R_k is the sequence made up of each k^{th} term taken from the end of each level beginning at the first level.

The first few left and right diagonals are mentioned below:
$L_1 = \left\{\frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \ldots\right\}, L_2 = \left\{\frac{2}{1}, \frac{2}{3}, \frac{2}{5}, \ldots\right\}, L_3 = \left\{\frac{3}{2}, \frac{3}{5}, \frac{3}{8}, \ldots\right\} \ldots$
$R_1 = \left\{\frac{1}{1}, \frac{2}{1}, \frac{3}{1}, \ldots\right\}, R_2 = \left\{\frac{1}{2}, \frac{3}{2}, \frac{5}{2}, \ldots\right\}, R_3 = \left\{\frac{2}{3}, \frac{5}{3}, \frac{8}{3}, \ldots\right) \ldots$

Note: If a particular level of the Stern-Brocot tree does not have a k^{th} term, we omit that level and begin from the first level which has the k^{th} term. For example, the left diagonal L_4 is obtained by taking the 4^{th}

term from each row begining at the first level, but since the first and second levels of the tree do not have a 4^{th} term, we begin with the 4^{th} term of the 3^{rd} level.

Notation: If $L_k = \{L_{k,1}, L_{k,2}, L_{k,3}, \ldots\}$ and $L_m = \{L_{m,1}, L_{m,2}, L_{m,3}, \ldots\}$ then the sum of L_k and L_m is denoted by $L_k \oplus L_m$ and represents the left diagonal $\{L_{k,1} \oplus L_{m,1}, L_{k,2} \oplus L_{m,2}, \ldots\}$.

Lemma 4.2. *[5] If $k = 2^i(2j+1)$ where $i, j \in N$, then*

i) $L_{2k} = \begin{cases} L_k \oplus L_{1^+}, & \text{if } j = 0 \\ L_k \oplus L_{j+1}, & \text{if } j > 0 \end{cases}$ *where* $L_{1^+} = \left\{\frac{1}{0}, \frac{1}{1}, \frac{1}{2}, \ldots\right\}$

ii) $L_{2k+1} = L_{k+1} \oplus L_{j+1}$.

Remark 4.1. *The goal of Lemma 4.2 is to show that any left diagonal in the Stern-Brocot tree is obtained by the mediant sum operation "$\oplus$" of some previous two left diagonals. For example, the left diagonal L_6 is obtained by $L_3 \oplus L_2$. This is the motivation for the Stern-Brocot trees that we construct corresponding to our epichristoffel trees, which is shown in Theorem 5.2,as it follows the same rules of construction. The only difference is instead of adding two fractions with the mediant operation we add two tuples with the mediant operation. Therefore, the left diagonals of the Stern-Brocot trees that correspond to our epichristoffel trees also share the same property that any left diagonal can be obtained by the mediant sum of some previous two left diagonals.*

Theorem 4.2. *[5] If $\frac{a}{b}$ is the k^{th} entry of the n^{th} row of the Stern-Brocot tree where $n > 1$, then the k^{th} entry of the $(n+1)th$ row is $\frac{a}{a+b}$.*

Theorem 4.3. *[5] If $\frac{a}{b}$ is the k^{th} entry of the n^{th} row of the Stern-Brocot tree where $n > 1$, then the k^{th} entry of the $(n+1)th$ row is $\frac{a+b}{b}$.*

From the Stern-Brocot tree we can thus deduce that if there exists a Christoffel word of slope $\frac{a}{b}$ then there will always exist a Christoffel word of slope $\frac{a}{b+na}$ and a Christoffel word of slope $\frac{a+nb}{b}$ for any natural number n. More results on the Stern-Brocot tree can be found in [4] and [5].

5 Epichristoffel Trees

In this section, we introduce infinite binary trees for epichristoffel words similar to the Christoffel tree.

Consider any c-epichristoffel word w over the ordered alphabet $A = \{a_0, a_1, ..., a_{k-1}\}$ of length $n > 1$. Since w can be nonuniquely written as the product of two c-epichristoffel words, as shown in Example 3.4. Consider such a factorization of (u, v) of w where u and v are c-epichristoffel.

Lemma 5.1. *If $w = uv$ is a c-epichristoffel word whose factorization is obtained as explained above, then the words $w_1 = uuv$ and $w_2 = uvv$ are also c-epichristoffel words.*

Proof. Since w is a c-epichristoffel word, it is obtained by the application of a sequence of episturmian morphisms to some letter, say $a_j \in A$, where $0 \leq j \leq k-1$.
Therefore,
$w = \psi_{a_{i_1}} \psi_{a_{i_2}} ... \psi_{a_{i_l}}(a_j)$, where $0 \leq i_1, i_2, ..., i_l \leq k-1$ and $i_l \neq j$.
$= \psi_{a_{i_1}} \psi_{a_{i_2}} ... \psi_{a_{i_{l-1}}}(a_{i_l} a_j)$
$= \psi_{a_{i_1}} \psi_{a_{i_2}} ... \psi_{a_{i_{l-1}}}(a_{i_l}) \psi_{a_{i_1}} \psi_{a_{i_2}} ... \psi_{a_{i_{l-1}}}(a_j)$
Thus, $w = uv$, where $u = \psi_{a_{i_1}} \psi_{a_{i_2}} ... \psi_{a_{i_{l-1}}}(a_{i_l})$ and $v = \psi_{a_{i_1}} \psi_{a_{i_2}} ... \psi_{a_{i_{l-1}}}(a_j)$
Consider now the word w_1 formed by the application of the following episturmian morphisms to the lettera_j:
$w_1 = \psi_{a_{i_1}} \psi_{a_{i_2}} ... \psi_{a_{i_{l-1}}} \psi_{a_j} \psi_{a_{i_l}}(a_j)$

$= \psi_{a_{i_1}} \psi_{a_{i_2}} \ldots \psi_{a_{i_{l-1}}} \psi_{a_j}(a_{i_l} a_j)$
$= \psi_{a_{i_1}} \psi_{a_{i_2}} \ldots \psi_{a_{i_{l-1}}} \psi_{a_j}(a_{i_l}) \psi_{a_{i_1}} \psi_{a_{i_2}} \ldots \psi_{a_{i_{l-1}}} \psi_{a_j}(a_j)$
$= \psi_{a_{i_1}} \psi_{a_{i_2}} \ldots \psi_{a_{i_{l-1}}} (a_j a_{i_l}) \psi_{a_{i_1}} \psi_{a_{i_2}} \ldots \psi_{a_{i_{l-1}}} (a_j)$
$= \psi_{a_{i_1}} \psi_{a_{i_2}} \ldots \psi_{a_{i_{l-1}}} (a_j) \psi_{a_{i_1}} \psi_{a_{i_2}} \ldots \psi_{a_{i_{l-1}}} (a_{i_l}) \psi_{a_{i_1}} \psi_{a_{i_2}} \ldots \psi_{a_{i_{l-1}}} (a_j)$
$= vuv$
$\equiv uvv$
Therefore, w_2 is also c-epichristoffel.
Similarly, it can be seen that w_1 is obtained by computing $\psi_{a_{i_1}} \psi_{a_{i_2}} \ldots \psi_{a_{i_l}} \psi_{a_{i_l}}(a_j)$ and is also c-epichristoffel.

□

5.1 The construction of an epichristoffel tree

We construct the epichristoffel tree as follows: Let w be an arbitrary epichristoffel word of length n corresponding to the k-tuple $(x_1, x_2, \ldots, x_k)$. The word w is obtained by applying a sequence of episturmian morphisms on a letter $a_j \in A$, after which the smallest word in its conjugacy class is considered. In the process of doing so we factorize w into two c-epichristoffel words say, u and v as shown in the proof of Lemma 5.1.
Let $(l_1, l_2, \ldots, l_k)$ and $(m_1, m_2, \ldots, m_k)$ denote the corresponding k-tuples for the words u and v respectively.

The epichristoffel word $w = w[1]w[2]\ldots w[n]$ is the smallest word in the conjugacy class obtained after the above episturmian morphims are applied. Due to this construction either $w[1]w[2]\ldots w[|u|]$ or $w[1]w[2]\ldots w[|v|]$ will be the epichristoffel word corresponding to u or v. We use the word w as the root of the tree. Suppose, without loss of generality that if $w[1]w[2]\ldots w[|v|]$ is the epichristoffel word corresponding to v, then the root of the tree is (u', v') where
$u' = w[1]w[2]\ldots w[|v|]$ and $v' = w[|v|+1]w[|v|+2]\ldots w[|v|+|u|]$
The left and right descendants of this tree, as in the case of the Christoffel tree are $(u', u'v')$ and $(u'v', v')$ respectively. These in turn are once again epichristoffel due to Lemma 5.1 and the fact that $u'v'$ is the smallest with respect to the lexicographic ordering. Suppose, if $w[1]w[2]\ldots w[|u|]$ is the epichristoffel word corresponding to u, then the root of the tree is (u', v') where $u' = w[1]w[2]\ldots w[|u|]$ and $v' = w[|u|+1]w[|u|+2]\ldots w[|u|+|v|]$ This construction is illustrated in the example below:

Example 5.1. *Let $A = \{x, y, z\}$. Consider the 3-tuple (1,2,4). To find u and v, we consider the episturmian morphisms applied to construct the word zyzzyzx.*
$(1,2,4) \to^z (1,2,1) \to^y (1,0,1) \to^z (1,0,0)$
$\psi_z\psi_y\psi_z(x) = \psi_z\psi_y(zx) = \psi_z\psi_y(z)\psi_z\psi_y(x) = \psi_z(yz)\psi_z(yx) = zyz\ zyzx$
Thus, $u = zyz$ and $v = zyzx$, whose corresponding tuples are (0,1,2) and (1,1,2). The epichristoffel word corresponding to zyzzyzx is $w = xzyzzyz$ and $w[1]w[2]w[3]w[4] = xzyz$ which is the epichristoffel word corresponding to the tuple (1,1,2).

The epichristoffel tree is obtained by considering w as the root with the factorization $(xzyz, zyz)$ and the left and right descendants of each node follows the rule mentioned above.

Similar to the Stern-Brocot tree that is isomorphic to the Christoffel tree via the map that associates to each Christoffel word $w = (u, v)$ to the fraction $\frac{|w|_a}{|w|_b}$, we construct an infinite binary tree of k-tuples that can be associated to the epichristoffel tree via the map that associates each c-epichristoffel word $w = (u, v)$ to the k-tuple $(x_1, x_2, \ldots, x_k)$ where $x_i = |w|_{a_i}$.

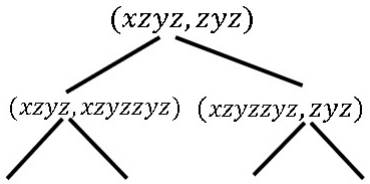

Figure 4: The epichristoffel tree with the root word $(xzyz, zyz)$

Definition 5.1. The mediant of two k-tuples $(y_1, y_2, \ldots, y_k)$ and $(z_1, z_2, \ldots, z_k)$ is the be k-tuple $(y_1 + z_1, y_2 + z_2, \ldots, y_k + z_k)$.
Let $(u'_1, u'_2, \ldots, u'_k)$ and $(v'_1, v'_2, \ldots, v'_k)$ be two k-tuples corresponding to the words u' and v'.
Define the sequence S_i as follows:
$S_0 = (u'_1, u'_2, \ldots, u'_k), (v'_1, v'_2, \ldots, v'_k)$ and
S_i for $i > 1$ is obtained from S_{i-1} by inserting between two consecutive elements of the sequence, their mediants. The mediants constructed in the *i-th* iteration of the above process for $i > 0$ are the vertices in the *i-th* level of this tree. We call this as the Stern-Brocot tree corresponding to the epichristoffel tree.

The Stern-Brocot sequence and tree corresponding to the epichristoffel tree of Example 5.1 is shown below:

$$S_0 = (1,1,2), (0,1,2)$$
$$S_1 = (1,1,2), (\mathbf{1},\mathbf{2},\mathbf{4}), (0,1,2)$$
$$S_2 = (1,1,,2), (\mathbf{2},\mathbf{3},\mathbf{6}), (1,2,4), (\mathbf{1},\mathbf{3},\mathbf{6}), (1,1,2)$$

and so on.

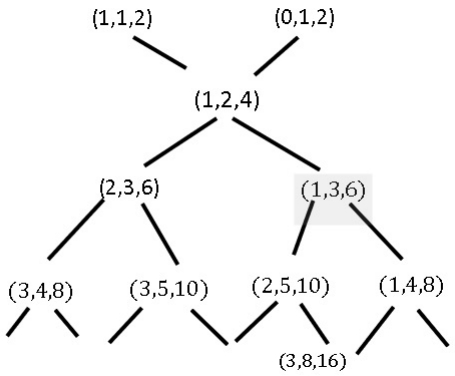

Figure 5: The Stern-Brocot tree for Example 5.1

Example 5.2. *Let $A = \{x, y, z\}$. Consider the 3-tuple (3,2,1). The c-epichristoffel word corresponding to this tuple is xy xyxz. Here $u = xy$ and $v = xyxz$, whose corresponding tuples are (1,1,0) and (2,1,1) respectively. Hence, the epichristoffel word corresponding to this word is itself, that is $w = xyxyxz$. The respective epichristoffel and Stern-Brocot trees are shown in Figures 7 and 8.*

Remark 5.1. *To construct an epichristoffel tree we first begin with an epichristoffel word of length n, which serves as the root of the tree. To obtain an epichristoffel word we apply a set of episturmian*

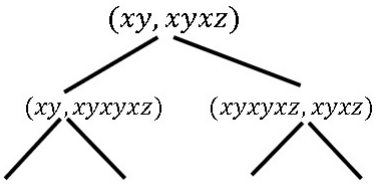

Figure 6: The epichristoffel tree with the root word $(xy, xyxz)$

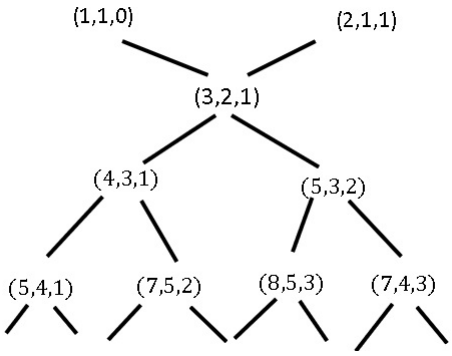

Figure 7: The Stern-Brocot tree for Example 5.2

morphisms to a letter. Once this is done, we may or may not have the epichristoffel word depending on whether the word is the smallest according to the lexicographic ordering in its conjugacy class. If the word obtained is not the smallest in its conjugacy class, we have obtained a c-epichristoffel word. This has been explained in Lemma 5.1. While applying the set of episturmian morphisms to the letter, we also factorize the word as shown in proof of Lemma 5.1 and write it as (u,v), adhering to the notation of the construction of the Christoffel tree. If $w = w[1]w[2]\dots w[n]$ is the smallest word in the conjugacy class of (u,v), then either $w[1]w[2]\dots w[|u|]$ or $w[1]w[2]\dots w[|v|]$ is the epichristoffel word corresponding to the c-epichristoffel word u or the c-epichristoffel word v. This has been illustrated in Example 5.1 and Example 5.2. If without loss of generality, $w[1]w[2]\dots w[|u|]$ is the epichristoffel word corresponding to the c-epichristoffel word u, then $w[u+1]w[u+2]\dots w[|u+v|]$ is the epichristoffel word corresponding to the c-epichristoffel word v. We now begin the construction of our epichristoffel tree taking (u',v') as the root of our tree where $u' = w[1]w[2]\dots w[|u|]$ and $v' = w[u+1]w[u+2]\dots w[|u+v|]$. The left and right descendants of this tree, as in the case of the Christoffel tree are $(u',u'v')$ and $(u'v',v')$ respectively. Since (u',v') is an epichristoffel word, it is the smallest word with respect to the lexicographic order in its conjugacy class. Therefore, $u' < v'$ in the lexicographic ordering. From which we can conclude that the word $(u',u'v')$ is also smallest in its conjugacy class as we have only added the epichristoffel word u' that is smaller as a prefix. Similarly, we can deal with the case of $(u'v',v')$ where we have added the epichristoffel word v' which is larger as a suffix. Lemma 5.1 has already established that whenever (u,v) is c-epichristoffel then (uv,v) and (u,uv) are also c-epichristoffel. Since epichristoffel words are words in the conjugacy class, they are c-epichristoffel. As they are the smallest in the conjugacy class, they are epichristoffel words.

Theorem 5.1. *Every word appearing in the epichristoffel tree is obtained by the application of a finite number of episturmian morphisms to a letter and is lexicographically smallest in its respective conjugacy*

class making it an epichristoffel word.

Proof. The root of the epichristoffel tree is a word $w = w[1]w[2]...w[n]$ of length n that is the smallest word in the conjugacy class of a word w' obtained by an application of a sequence of episturmian morphisms as shown in the construction and is thus an epichristoffel word.
This root word w is factorized as
$w[1]w[2]...w[|u|], w[|u|+1]w[|u|+2]...w[|u|+|v|]$ or $w[1]w[2]...w[|v|], w[|v|+1]w[v+2]...w[|v|+|u|]$
where $|u|+|v| = n$, depending on whether $w[1]w[2]...w[|u|]$ is the epichristoffel word corresponding to the word u or $w[1]w[2]...w[|v|]$ is the epichristoffel word corresponding to the word v respectively, where u, v is the factorisation of w' as the product of two $c-$ epichristoffel words as shown in Example 3.3.
Without loss of generality, if $w[1]w[2]...w[|u|], w[|u|+1]w[|u|+2]...w[|u|+|v|]$ is the factorization of w then the left and right descendants in the tree, respectively, are
$w[1]w[2]...w[|u|], w[1]w[2]...w[|u|]w[|u|+1]w[|u|+2]...w[|u|+|v|]$ and
$w[1]w[2]...w[|u|]w[|u|+1]w[|u|+2]...w[|u|+|v|], w[|u|+1]w[|u|+2]...w[|u|+|v|]$.
Since $w[1]w[2]...w[n]$ is the smallest in its conjugacy class, we have
$w[1]w[2]...w[|u|] < w[|u|+1]w[|u|+2]...w[|u|+|v|]$.
Thus, the left and right descendants are also the smallest in their respective conjugacy classes and by Lemma 5.1 they are $c-$ epichristoffel due to which these descendants are also epichristoffel words. This pattern continues for all the words appearing in the epichristoffel tree and thus every word in this tree is an epichristoffel word.

□

Thus, using an arbitrary epichristoffel word, the epichristoffel tree can be used to determine the existence of epichristoffel words of various lengths. Although a characterization of the epichristoffel word for each conjugacy class was not found in [7], the epichristoffel tree can give us the characterization of an epichristoffel word corresponding to a tuple present in the tree. This is illustrated below:

Example 5.3. *To determine the epichristoffel word in the conjugacy class of the c-epichristoffel word of the tuple,* $(3,8,16)$ *we need only to trace the path from the root of the epichristoffel tree to the node that represents the tuple (3,8,16). The root of this tree, as seen in Example 5.1 is the word (xzyz,zyz). To find the word corresponding to the tuple (3,8,16), we must proceed to the right child given by the node,* $(xzyzzyz, zyz)$ *followed by its left child given by the node* $(xzyzzyz, xzyzzyzzyz)$*, which is followed by its right child given by the node* $(xzyzzyzxzyzzyzzyz, xzyzzyzzyz)$*. Thus, the epichristoffel for the conjugacy class of the epichristoffel tuple (3,8,16) is given by the word xzyzzyzxzyzzyzzyzxzyzzyzzyz.*

Remark 5.2. *We are using the epichristoffel tree to give a characterization of each epichristoffel word of its conjugacy class that appears on the tree as this was the question raised by Paquin in [7]. For example, the c-epichristoffel word corresponding to the tuple (3,8,16) is determined by applying the episturmian morphisms* $\psi_z\psi_y\psi_z\psi_x\psi_z\psi_z$ *to the letter x since, (3,8,16)* $\to_z$ *(3,8,5)* $\to_y$ *(3,0,5)* $\to_z$ *(3,0,2)* $\to_x$ *(1,0,2)* $\to_z$ *(1,01)* $\to_z$ *(1,0,0) . Doing so we obtain* $\psi_z\psi_y\psi_z\psi_x\psi_z\psi_z(x) = \psi_z\psi_y\psi_z\psi_x\psi_z(zx) = \psi_z\psi_y\psi_z\psi_x(zzx) = \psi_z\psi_y\psi_z(xzxzx) = \psi_z\psi_y(zxzzxzzx) = \psi_z(yzyxyzyzyxyzyzyx) = zyzzyzxzyzzyzzyzxzyzzyzzyzx$*. The question raised by Paquin was to determine, given any c-epichristoffel word, such as the one above, a characterization of the epichristoffel word in its conjugacy class. We have shown that by locating the word on the epichristoffel tree, we receive the desired word in the conjugacy class. This has been explained in Example 5.3. Thus, by constructing an epichristoffel tree and its corresponding Stern-Brocot tree, we have successfully determined the epichristoffel word in the conjugacy class of each word on the tree. In Example 5.3, the tree gives us a characterization of the epichristoffel word in the conjugacy class of all the tuples occurring in it such as (2,3,6),(1,3,6),(3,4,8),(3,5,10),(2,5,10),(1,4,8),(3,8,16) etc.*

The following result is obtained by ordering the epichristoffel tree using the definition of left and right diagonals of Section 4.

Theorem 5.2. *If w is an epichristoffel word corresponding to the tuple $(x_1, x_2, \ldots x_k)$, then there exist epichristoffel words for tuples of the form $(x_1 + d_1, x_2 + d_2, \ldots x_k + d_k)$, for some positive integers $d_1, d_2, \ldots, d_k$.*

Proof. Since the mediant operation used for two fractions in the Stern-Brocot tree is extended for the case of k-tuples, Lemma 4.2 can be applied analogously for the case of the epichristoffel tree.
Hence, each left diagonal of the Stern-Brocot tree corresponding to the epichristoffel tree is the sum of some previous left diagonals of the tree. Now, if L and L' are any two left diagonals that satisfy the hypothesis then their sum also satisfies the hypothesis. Since
$L = \{(x_1, x_2, \ldots, x_k), (x_1 + d_1, x_2 + d_2, \ldots x_k + d_k), (x_1 + 2d_1, x_2 + 2d_2, \ldots x_k + 2d_k), \ldots\}$ and
$L' = \{(y_1, y_2, \ldots, y_k), (y_1 + e_1, y_2 + e_2, \ldots y_k + e_k), (y_1 + 2e_1, y_2 + 2e_2, \ldots y_k + 2e_k), \ldots\}$,
then,
$L \oplus L' = \{(x_1 + y_1, x_2 + y_2, \ldots, x_k + y_k), (x_1 + y_1 + d_1 + e_1, x_2 + y_2 + d_2 + e_2, \ldots, x_k + y_k + d_k + e_k), \ldots\}$
Thus, each left diagonal of the Stern-Brocot tree corresponding to an epichristoffel tree satisfies the result.

□

Remark 5.3. *The Stern-Brocot trees corresponding to the epichristoffel trees follows the same construction but for tuples rather than fractions. That is, $(a,b,c) \oplus (d,e,f) = (a+d, b+e, c+f)$. Therefore, the result can be extended analogously.*

Remark 5.4. *Theorem 5.2, is an extension of Lemma 4.2 for the case of tuples, that is, every left diagonal is the mediant sum of some previous two left diagonals. We are using this to show the existence of epichristoffel words of various lengths in the latter part of the paper. We use L and L' as an example to show that if they are any two left diagonals that satisfy the hypothesis then their mediant sum also satisfies the hypothesis. Since every left diagonal is the mediant sum of previous two left diagonals and if those two left diagonals satisfy the hypothesis then their sum must also satisfy the hypothesis.*

The above result helps to answer the question raised by Paquin in [7] on the existence of epichristoffel words of various lengths. This is illustrated in the example below.

Example 5.4. *The first three left diagonals of the epichristoffel tree in Example 5.1 produce epichristoffel words of lengths $4n+3, 8n+2$ and $12n+5$ for $n = 1, 2, 3\ldots$*

$$L_1 = \{(1,2,4), (2,3,6), \ldots, (n, n+1, 2n+2), \ldots\}$$
$$L_2 = \{(1,3,6), (3,5,10), \ldots, (2n-1, 2n+1, 4n+2), \ldots\}$$
$$L_3 = \{(2,5,10), (5,8,16), \ldots, (3n-1, 3n+2, 6n+4), \ldots\}$$

The first three right diagonals of the epichristoffel tree in Example 5.1 produce epichristoffel words of lengths $3n+4, 6n+5$ and $9n+9$ for $n = 1, 2, 3\ldots$

$$R_1 = \{(1,2,4), (1,3,6), \ldots, (1, n+1, 2n+2), \ldots\}$$
$$R_2 = \{(2,3,6), (2,5,10), \ldots, (2, 2n+1, 4n+2), \ldots\}$$
$$R_3 = \{(3,5,10), (3,8,16), \ldots, (3, 3n+2, 6n+4), \ldots\}$$

Remark 5.5. *The computation of the left diagonals $L_1, L_2, L_3, \ldots$ and the right diagonals $R_1, R_2, R_3, \ldots$ follow from Definition 4.4. Consider the Stern-Brocot tree in Example 5.1. Here L_1 denotes the 1st term from each level beginning at the first level, that is, $(1,2,4),(2,3,6),(3,4,8),\ldots$. L_2 denotes the 2nd term from each level. Since the first level has only one term, we begin at the next level, that is, $(1,3,6),(3,5,10),(5,7,14),\ldots$ and so on. R_1 denotes the 1st term taken from the end of each level, that is $(1,2,4),(1,3,6),(1,4,8),\ldots$. R_2 denotes the 2nd term taken from the end of each level, that is, $(2,3,6),(2,5,10),\ldots$ and so on.*

Theorem 5.3. *For a 3-letter alphabet, $A = \{x,y,z\}$, there exist epichristoffel words having occurences of each letter at least once of every length $n \geq 4$ except for $n = 5$.*

Proof. We first begin by observing that for any given even number $n \geq 4$, one can determine an epichristoffel word by simply considering a tuple of the form $(1,1,2m)$ where $m = 1,2,3,\ldots$
It is easy to see that a tuple of the above form satisfies the condition of Proposition 3.1.
The right diagonal R_1 in Example 5.4 shows the existence of epichristoffel words of length $3n+4$. Since we have already established the existence of words of even length, we can ignore the words of even length from words with length of the form $3n+4$ obtaining words with lengths $7,13,19,\ldots,6n+1$, for $n = 1,2,3,\ldots$
Constructing an epichristoffel tree with the root node as the word (xzz,yzz) which corresponds the tuple (1, 1, 4), the left-diagonal L_1 yields tuples of the form $(n,1,2n+2)$ that guarantees the existence of epichristoffel words of length, $3n+3$ from which we can conclude that there exist epichristoffel words of length $6n+3$.
The right diagonal R_2 in Example 5.4 yields epichristoffel words of length $6n+5$. Thus, the existence of epichristoffel words of lengths $6n+1, 6n+3, 6n+5$ are established and since any epichristoffel word of even length also exists, the result is established. □

Another way that Christoffel and epichristoffel words differ from one another is that, as seen in Lemma 3.4, epichristoffel words are not necessarily factorizable as a product of two epichristoffel words, in contrast to Christoffel words, which can always be factorized as a product of two Christoffel words. The epichristoffel tree can be used to identify some subclasses of epichristoffel words that can or cannot be factorized as a product of epichristoffel words, as we demonstrate below.

To see this, we start with an arbitrary epichristoffel word that cannot be factorized as the product of two other epichristoffel words. For instance, the epichristoffel word *xzyzzyz*, which appears as the root of the epichristoffel tree with the factorization $(xzyz,zyz)$, corresponding to the tuple (1,2,4). Here, the word *zyz* is not epichristoffel. Consider the path that starts at the tree's root and descends to its right child, namely $(xzyzzyz,zyz)$. Concatenating the word *zyz* to the word *xzyzzyz* ensures that the word is still not factorizable as two epichristoffel words since *zyz* is not an epichristoffel word. In a similar vein, the epichristoffel word $(xzyzzyz,zyz)$, when we take into consideration has its right child to be $(xzyzzyzzyz,zyz)$, which likewise cannot be factorized as the product of two epichristoffel words. As a result, a subclass of epichristoffel words that lack the ability to be factorized as the product of two epichristoffel words can be identified with the aid of the epichristoffel tree.

Continuing with the above example it is seen that the set:

$\{(1,2,4),(1,3,6),(1,4,8),\ldots(1,n+1,2n+2),\ldots\}$ has examples of tuples that cannot yield the product of two epichristoffel words. These tuples are precisely the right diagonal R_1 of the epichristoffel tree in Figure 5.

On the other hand, if we begin with a word that can be factorized as the product of two epichristoffel words, then all its descendants on the epichristoffel tree will possess a factorization as two epichristoffel

words since if (u,v) is such a node, with both u and v as epichristoffel, then (u,uv) and (uv,v) are factorizations of epichristoffel words as a product of epichristoffel words. We thus get the following results:

Theorem 5.4. *If w is an epichristoffel word that cannot be factorized as the product of two epichristoffel words then the right diagonal R_1 of the epichristoffel tree formed using w as the root contains epichristoffel words that cannot be factorized as the product of two epichristoffel words.*

Proof. Since w cannot be factorized as the product of two epichristoffel words, constructing the epichristoffel word using w gives us the root node of the form (u,v) where v is not an epichristoffel word. The right diagonal R_1 will then consist of words of the form $(u,v),(uv,v),(uvv,v)...$ and thus provides us with an infinite subclass of epichristoffel words that cannot be factorized as a product of two epichristoffel words. □

Theorem 5.5. *If w is an epichristoffel word that can be factorized as the product of two epichristoffel words then every word of the epichristoffel tree formed using w as the root can be factorized as the product of two epichristoffel words.*

Proof. If w is an epichristoffel word that can be factorized as the product of two epichristoffel words then it appears in the epichristoffel tree in the form (u,v) where u and v are both epichristoffel words. The left and right descendants respectively are (u,uv) and (uv,v) and thus possess a factorization as the product of two epichristoffel words. □

Remark 5.6. *A related question is the following: Given a tuple, is there a unique c-epichristoffel word associated to it? Proposition 3.1 gives us the method to determine for a given tuple, if there exists a c-epichristoffel word associated to it. This word is unique up to conjugation or circular shifts as the method of applying the iterations may vary. For example, the tuple (1,2,4) is a tuple corresponding to a c-epichristoffel word. The c-epichristoffel word can be obtained by applying the episturmian morphisms $\psi_z\psi_y\psi_z$ to the letter x or the episturmian morphisms $\psi_z\psi_y\psi_x$ to the letter z. The first case yields the word zyzzyzx, the second case yields the word zyzxzyz. Although they are different, they are both conjugate to each other, that is, one can be obtained from the other by circular shifts of the letters.*

6 Conclusion

In this paper, we have continued Paquin's study in [7] on epichristoffel words by extending the definition of the Christoffel tree to accommodate epichristoffel words. In doing so, we can answer some of the questions raised, such as the existence of epichristoffel words of various lengths and provide a characterization for the conjugacy class of every word appearing on the tree. The epichristoffel tree can be a useful tool in determining epichristoffel words that possess a factorization property as smaller epichristoffel words and those that do not. Further studying these trees may help provide insights into the Fraenkel conjecture, which states that there exists a unique infinite word up to letter permutation and conjugation that is balanced and has pair-wise distinct letter frequencies over a k-letter alphabet, which is periodic and of the form $ppp\ldots$ where p is an epichristoffel word.

References

[1] V. Berthé, A. de Luca & C. Reutenauer (2007): *On an involution of Christoffel words and Sturmian morphisms*. European Journal of Combinatorics 29, pp. 535–553, doi:10.1016/j.ejc.2007.03.001.

[2] J.-P. Borel & F. Laubie (1993): *Quelques mots sur la droite projective réelle*. Journal De Théorie Des Nombres De Bordeaux 5, p. 2351, doi:10.5802/jtnb.77.

[3] J.-P. Borel & C. Reutenauer (2006): *On Christoffel classes*. RAIRO - Theoretical Informatics and Applications 40, pp. 15–28, doi:10.1051/ita:2005038.

[4] Bates Bruce (2010): *Linking the Calkin - Wilf and Stern-Brocot Trees*. European Journal of Combinatorics 31(7), pp. 1637–1661, doi:10.1016/j.ejc.2010.04.002.

[5] Bates Bruce (2010): *Locating Terms in the Stern-Brocot Tree*. European Journal of Combinatorics 31(3), pp. 1020–1033, doi:10.1016/j.ejc.2007.10.005.

[6] E.B. Christoffel (1873): *Observatio arithmetica*. Annali Di Matematica Pura Ed Applicat 6, pp. 148–152, doi:10.1007/BF02420125.

[7] Paquin Genevieve (2009): *On a generalization of Christoffel words: epichristoffel words*. Theoretical Computer Science 410(38), pp. 3782–3791, doi:10.1016/j.tcs.2009.05.014.

[8] G. Pirillo J. Justin (2002): *Episturmian words and episturmian morphisms*. Theoretical Computer Science 276, pp. 281–313, doi:10.1016/S0304-3975(01)00207-9.

[9] Berstel Jean, Aaron Lauve, Christophe Reutenauer & Franco Saliola (2008): *Combinatorics on Words: Christoffel Words and Repetitions in Words*. American Mathematical Society.

[10] C. Kassel & C. Reutenauer (2007): *Sturmian morphisms, the braid group B4, Christoffel words and bases of F2*. Annali di Matematica 186, pp. 317–339, doi:10.1007/s10231-006-0008-z.

[11] M. Lothaire (2002): *Algebraic Combinatorics on Words*. Cambridge University Press, doi:10.1017/CBO9781107326019.

[12] A. de Luca & F. Mignosi (1994): *Some combinatorial properties of Sturmian words*. Theoretical Computer Science 136, pp. 361–385, doi:10.1016/0304-3975(94)00035-H.

[13] Christophe Reutenauer (2018): *From Christoffel Words to Markoff Numbers*. Oxford University Press, doi:10.1093/oso/9780198827542.001.0001.

[14] Lama Tarsissi (2017): *Balance properties on Christoffel words and applications*. General Mathematics [math.GM], Universite Grenoble Alpes.

Input-Driven Pushdown Automata with Translucent Input Letters

Martin Kutrib Andreas Malcher
Matthias Wendlandt
Institut für Informatik, Universität Giessen
Arndtstr. 2, 35392 Giessen, Germany
{kutrib, andreas.malcher, matthias.wendlandt}@informatik.uni-giessen.de

Input-driven pushdown automata with translucent input letters are investigated. Here, the use of translucent input letters means that the input is processed in several sweeps and that, depending on the current state of the automaton, some input symbols are visible and can be processed, whereas some other symbols are invisible, and may be processed in another sweep. Additionally, the returning mode as well as the non-returning mode are considered, where in the former mode a new sweep must start after processing a visible input symbol. Input-driven pushdown automata differ from traditional pushdown automata by the fact that the actions on the pushdown store (push, pop, nothing) are dictated by the input symbols. We obtain the result that the input-driven nondeterministic model is computationally stronger than the deterministic model both in the returning mode and in the non-returning mode, whereas it is known that the deterministic and the nondeterministic model are equivalent for input-driven pushdown automata without translucency. It also turns out that the non-returning model is computationally stronger than the returning model both in the deterministic and nondeterministic case. Furthermore, we investigate the closure properties of the language families introduced under the Boolean operations. We obtain a complete picture in the deterministic case, whereas in the nondeterministic case the language families are shown to be not closed under complementation. Finally, we look at decidability questions and obtain the non-semidecidability of the questions of universality, inclusion, equivalence, and regularity in the nondeterministic case.

1 Introduction

The usual way of processing input on language recognition devices is by reading input strings from left to right, one symbol at a time, and finally providing an accept or reject decision when arriving at the end of the input. This "standard" input mode has been extended in the literature in several directions. For example, two-way motion of the input head, stationary moves of the input head, rotating the input head, or restarting modes have been considered for a wide variety of machines (the reader is referred to, e.g., [2, 6, 7, 23] for an overview of these and other models). In all these cases, the extensions have consequences on the computational and descriptional power of the machines. Thus, the way of processing the input can be considered as a computational resource that then can be used to tune computational and descriptional power.

Of particular interest in recent literature have been *discontinuous* ways of input processing, where one of them is the "jumping" paradigm which means that jumping to any position inside the input string is allowed at any move. This paradigm has been investigated for finite automata in [11], where it is shown that this discontinuous input processing may increase the computational power since some non-context-free languages can be accepted. On the other hand, the discontinuous input mode may also limit the computational power since some regular languages cannot be accepted. A restricted variant, called "right one-way jumping" automata, has been considered in [1, 4]. Another way to discontinuously

N. Moreira and L. Prigioniero (Eds.): 15th International Workshop on
Non-Classical Models of Automata and Applications (NCMA 2025).
EPTCS 422, 2025, pp. 29–43, doi:10.4204/EPTCS.422.3

process the input is to use *translucent letters*. This concept has been introduced by Nagy and Otto in [15] for deterministic and nondeterministic finite automata and the basic idea for translucent devices is to provide a translucency function that defines in which states which letters of the input are translucent. At each move, the device skips (by looking through) the translucent portion of the input, from the current input head position up to the first non-translucent letter (thus realizing a jump). After processing the non-translucent symbol, in the *returning mode* the input head returns to the left end of the input while, in the *non-returning mode*, the input head continues to process the input according to its updated current state and the corresponding translucent symbols. In both modes, the input head returns to the left end when the right end of the input is reached.

Deterministic and nondeterministic finite automata with translucent letters have deeply been investigated in the literature (see, e.g., [13, 16, 22]). However, many questions are still open. Some recently studied variants are finite automata with translucent words [19], finite automata with translucent letters and two-way motion of the input head [8], and returning and non-returning deterministic pushdown automata with translucent letters [9, 10]. It should be noted that returning pushdown automata with translucent letters have been studied by Nagy and Otto in [14, 17] in terms of certain cooperating distributed systems of restarting automata with additional pushdown store. However, in their model λ-transitions are not allowed and acceptance is defined by empty pushdown. In [21] the paradigm of *input-driven* languages was imposed to certain cooperating distributed systems of restarting automata with additional pushdown store with regard to characterizing certain trace languages. In input-driven pushdown automata [3, 12] the next action on the pushdown store (push, pop, nothing) is solely governed by the input symbol and to this end the input alphabet is split into three subsets. Input-driven pushdown automata possess nice features (see, e.g., the survey given in [20]) such as the equivalence of nondeterministic and deterministic models, the positive decidability of the inclusion problem and the positive closure under union, intersection, complementation, concatenation, and iteration. It should be noted that the positive decidability result as well as the positive closures only hold if both given automata have a compatible splitting of their input alphabets. In this paper, we study input-driven deterministic and nondeterministic pushdown automata with translucent letters both in the returning and non-returning mode and, therefore, extend the results from [21]. We also study the closure properties of the four corresponding language families under the Boolean operations as well as the status of their decidability questions.

The paper is organized as follows. After giving the necessary definitions and two illustrating examples in the next section, we study in Section 3 the impact of nondeterminism in our models and it turns out that the nondeterministic model is computationally stronger in the returning mode as well as in the non-returning mode. In Section 4 we compare returning and non-returning models. We yield proper inclusions of the returning language families in the non-returning language families in the deterministic as well as in the nondeterministic case. Moreover, the combination deterministic and non-returning versus the combination nondeterministic and returning leads to incomparability results. The closure under the Boolean operations is studied in Section 5 and we obtain the closure under complementation, but non-closure under union and intersection in the returning and non-returning deterministic case. In the nondeterministic case, we obtain non-closure under complementation for both variants and non-closure under intersection (even with regular languages) in the returning case. Finally, we consider the usually studied decidability questions in Section 6. In the returning case we have the decidability of the emptiness problem as well as of the finiteness problem both in the nondeterministic and deterministic case. In addition, universality is decidable in the deterministic case. On the other hand, we get the non-semidecidability of the questions of universality, equivalence, inclusion, and regularity in case of input-driven nondeterministic pushdown automata with translucent letters working in the returning as well as in the non-returning mode. This extends some results partially known for nondeterministic finite

automata with translucent input letters working in the returning mode (see [16]).

2 Definitions and Preliminaries

We denote by Σ^* the set of all words on the finite alphabet Σ, including the *empty word* λ, and let $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. For any word $w \in \Sigma^*$, we let $|w|$ denote its *length*, w^R its *reversal*, and $|w|_a$ the *number of occurrences* of the symbol $a \in \Sigma$ in w. We use $\subseteq$ for *inclusions*, and $\subset$ for *proper inclusion*. Given a set S, we denote by 2^S its *power set*, and by $|S|$ its *cardinality*. We write S_x to denote the set $S \cup \{x\}$, for a given element $x \notin S$. A *language* on Σ is any subset $L \subseteq \Sigma^*$. The *complement* of L is the language $\overline{L} = \Sigma^* \setminus L$, its reversal is $L^R = \{ w^R \mid w \in L \}$. The *shuffle* $L_1 \sqcup\!\sqcup L_2$ of two languages $L_1, L_2 \subseteq \Sigma^*$ is defined as $L_1 \sqcup\!\sqcup L_2 = \{ x_1 y_1 x_2 y_2 \cdots x_n y_n \mid x_1 x_2 \cdots x_n \in L_1, y_1 y_2 \cdots y_n \in L_2 \text{ with } x_i, y_i \in \Sigma^* \text{ and } 1 \leq i \leq n \}$. Let $\Sigma = \{a_1, a_2, \ldots, a_n\}$ and $\Psi : \Sigma^* \to \mathbb{N}_0^n$ be a mapping such that $\Psi(w) = (|w|_{a_1}, |w|_{a_2}, \ldots, |w|_{a_n})$. Let $L \subseteq \Sigma^*$ be a language. Then, $\Psi(L) = \{\Psi(w) \mid w \in L\}$ is the *Parikh image* of L. We say that two languages $L_1, L_2 \subseteq \Sigma^*$ are *letter-equivalent* if $\Psi(L_1) = \Psi(L_2)$. Two language families $\mathscr{L}_1$ and $\mathscr{L}_2$ are said to be *incomparable* whenever $\mathscr{L}_1$ is not a subset of $\mathscr{L}_2$ and vice versa.

A classical pushdown automaton is called input-driven if the current input symbol defines the next action on the pushdown store, that is, pushing a symbol onto the pushdown store, popping a symbol from the pushdown store, or changing the state without modifying the pushdown store. To this end, the input alphabet Σ is partitioned into the sets Σ_D, Σ_R, and Σ_N, that control the actions push (D), pop (R), and state change only (N).

Input-driven pushdown automata with translucent input letters are extensions of input-driven pushdown automata that do not necessarily have to read the current input symbol. Instead, depending on the current state of such devices, some of the input letters may be translucent (invisible). Accordingly, an input-driven pushdown automaton with translucent input letters either reads and processes (by deleting, if not the endmarker) the first visible input letter.

Formally, an *input-driven pushdown automaton with translucent input letters* (NIDPDAwtl) is a system $M = \langle Q, \Sigma, \Gamma, q_0, \triangleleft, \bot, \tau, \delta_D, \delta_R, \delta_N \rangle$, where Q is the finite set of *internal states*, Σ is the finite set of *input symbols* partitioned into the sets Σ_D, Σ_R, and Σ_N, with $\Sigma \cap Q = \emptyset$, Γ is the finite set of *pushdown symbols*, $q_0 \in Q$ is the *initial state*, $\triangleleft \notin \Sigma$ is the endmarker, $\bot \notin \Gamma$ is the *bottom-of-pushdown symbol*, $\tau \colon Q \to 2^\Sigma$ is the translucency mapping, and δ_D is the partial transition function mapping $Q \times \Sigma_D \times (\Gamma \cup \{\bot\})$ to $2^{Q \times \Gamma} \cup \{\texttt{accept}\}$, δ_R is the partial transition function mapping $Q \times \Sigma_R \times (\Gamma \cup \{\bot\})$ to $2^Q \cup \{\texttt{accept}\}$, and δ_N is the partial transition function mapping $Q \times (\Sigma_N \cup \{\triangleleft\}) \times (\Gamma \cup \{\bot\})$ to $2^Q \cup \{\texttt{accept}\}$.

The translucency mapping τ bears the following meaning: for any state $q \in Q$, the letters from the set $\tau(q)$ are *translucent (invisible) for* q, that is, whenever in q, the automaton M does not see these letters (or equivalently, M sees through such letters).

A *configuration* of M is a pair $(qw\triangleleft, \gamma)$ or accept, where $q \in Q$ is the current state, $w \in \Sigma^*$ is the part of the input left to be processed, and $\gamma \in \Gamma^* \bot$ denotes the current pushdown content, the leftmost symbol being the top of the pushdown store. The *initial configuration* for an input w is set to $(q_0 w \triangleleft, \bot)$.

Being in some configuration $(qw\triangleleft, z\gamma)$ with $z \in \Gamma \cup \{\bot\}$ and $z\gamma \in \Gamma^* \bot$, first M determines the next symbol to scan. Precisely, if $w\triangleleft = xy\triangleleft$ with $x \in \tau(q)^*$, $y \in \Sigma^*$, and $a \notin \tau(q)$ is the first symbol of $y\triangleleft$, then M processes a. One step from a configuration to its successor configuration is denoted by $\vdash$.

Let $q, q' \in Q$, $x \in \tau(q)^*$, $a \notin \tau(q)$, $y \in \Sigma^*$, and $z \in \Gamma$, $\gamma \in \Gamma^* \bot$. We set

1. $(qxay\triangleleft, z\gamma) \vdash (q'xy\triangleleft, z'z\gamma)$, if $a \in \Sigma_D$ and $(q', z') \in \delta_D(q, a, z)$,
2. $(qxay\triangleleft, \bot) \vdash (q'xy\triangleleft, z'\bot)$, if $a \in \Sigma_D$ and $(q', z') \in \delta_D(q, a, \bot)$,

3. $(qxay\lhd, z\gamma) \vdash (q'xy\lhd, \gamma)$, if $a \in \Sigma_R$ and $q' \in \delta_R(q,a,z)$,
4. $(qxay\lhd, \bot) \vdash (q'xy\lhd, \bot)$, if $a \in \Sigma_R$ and $q' \in \delta_R(q,a,\bot)$,
5. $(qxay\lhd, z\gamma) \vdash (q'xy\lhd, z\gamma)$, if $a \in \Sigma_N$ and $q' \in \delta_N(q,a,z)$,
6. $(qxay\lhd, \bot) \vdash (q'xy\lhd, \bot)$, if $a \in \Sigma_N$ and $q' \in \delta_N(q,a,\bot)$,
7. $(qx\lhd, z\gamma) \vdash (q'x\lhd, z\gamma)$, if $q' \in \delta_N(q,\lhd,z)$,
8. $(qx\lhd, \bot) \vdash (q'x\lhd, \bot)$, if $q' \in \delta_N(q,\lhd,\bot)$.

In addition, whenever, the transition function yields `accept` for the current configuration, the successor configuration is `accept`.

So, on the endmarker only δ_N is defined. Whenever the pushdown store is empty, the successor configuration is computed by the transition functions with the special bottom-of-pushdown symbol $\bot$ which is never removed from the pushdown. As usual, we define the reflexive and transitive closure of $\vdash$ by $\vdash^*$.

An input-driven pushdown automaton with translucent letters is said to be deterministic (DIDPDAwtl) if $|\delta_x(q,a,z)| \leq 1$ for $x \in \{D,N,R\}$ and all $q \in Q$, $a \in \Sigma_x$, and $z \in \Gamma \cup \{\bot\}$.

A word w is accepted by M if there is a computation on input w that ends with `accept`. The *language accepted* by M is $L(M) = \{ w \in \Sigma^* \mid w \text{ is accepted by } M \}$. In general, the family of all languages accepted by automata of some type X will be denoted by $\mathscr{L}(\mathsf{X})$.

Some properties of language families implied by classes of input-driven pushdown automata may depend on whether all automata involved share the same partition of the input alphabet. For easier writing, we call the partition of an input alphabet a *signature*.

In order to clarify these notions, we continue with an example.

Example 1. Let $\Sigma = \{a,b,\#\}$ be an alphabet. The language $L_{rep} \subseteq \Sigma^*$ is defined as $L_{rep} = \{ b^n(\#b^n)^k \sqcup a^n \mid n \geq 1, k \geq 0 \}$. Its complement $\overline{L}_{rep}$ belongs to the family $\mathscr{L}(\mathsf{NIDPDAwtl})$. It can be represented as union $L \cup L'$ where L' is the complement of the regular language $\{ b^+(\#b^+)^k \sqcup a^+ \mid k \geq 0 \}$ with respect to Σ, and

$$L = \{ b^{n_1}\#b^{n_2}\#\cdots\#b^{n_k} \sqcup a^n \mid k \geq 0, n, n_i \geq 1 \text{ for } 1 \leq i \leq k, \text{ and there exists } 1 \leq i \leq k \text{ such that } n_i \neq n \}.$$

An NIDPDAwtl accepting $\overline{L}_{rep}$ can initially guess whether it wants to accept L or L'. Since L' is regular it is accepted by some NIDPDAwtl that does not utilize its pushdow store. So, it does not care about what actually happens on the pushdown store. This means that it may have an arbitrary signature.

Now, we construct an NIDPDAwtl $M = \langle Q, \Sigma, \Gamma, q_0, \lhd, \bot, \tau, \delta_D, \delta_R, \delta_N \rangle$ accepting L as follows.

- $Q = \{q_0, q_1, p, r\}$,
- $\Sigma_D = \{b\}$, $\Sigma_R = \{a\}$, $\Sigma_N = \{\#\}$,
- $\Gamma = \{\bullet, B\}$,
- $\tau(q_0) = \tau(q_1) = \tau(p) = \{a\}, \quad \tau(r) = \{\#, b\}$.

In a first phase, M reads the input symbols b and # from left to right with symbols a translucent. At the beginning and whenever a # is read, M guesses whether the length of the next b-block has to be matched with the number of a's in the input. If not, M scans the b-block and pushes $\bullet$'s in state q_1. If yes, M enters state p and scans the b-block as well but now pushing B's. If M never guesses yes, the computation is rejected on the endmarker. Otherwise, on reading the next # or the endmarker, M enters state r. In this situation, the number of B's on top of the pushdown corresponds to the length of the

b-block guessed to be matched. Finally, for state r the symbols b and # are translucent. Now M reads the a's from left to right. For each a read, one B is popped. If and only if there are more a's than B's or vice versa then M accepts. So, we set:

$$
\begin{array}{lll}
(1)\ \delta_D(q_0,b,\bot) = \{(q_1,\bullet),(p,B)\}, & & (7)\ \delta_N(p,\lhd,B) = \{r\},\\
(2)\ \delta_D(q_1,b,\bullet) = \{(q_1,\bullet)\}, & & (8)\ \delta_R(r,a,B) = \{r\},\\
(3)\ \delta_N(q_1,\#,\bullet) = \{q_1,p\}, & & (9)\ \delta_R(r,a,\bullet) = \texttt{accept},\\
(4)\ \delta_D(p,b,\bullet) = \{(p,B)\}, & & (10)\ \delta_R(r,a,\bot) = \texttt{accept},\\
(5)\ \delta_D(p,b,B) = \{(p,B)\}, & & (11)\ \delta_R(r,\lhd,B) = \texttt{accept}.\\
(6)\ \delta_N(p,\#,B) = \{r\}, & &
\end{array}
$$

■

Deterministic pushdown automata with translucent letters have been defined in [9] also for the *non-returning* mode. In this mode, after a visible letter is processed, the input head does not return to the left end of the input, but it continues reading from the position of the visible letter just processed. Whenever the endmarker is reached and the transition on the endmarker yields a new state, the computation is continued in this state, with the input head placed at the left end of the remaining input. Such deterministic pushdown automata with translucent letters working in the non-returning mode generalize non-returning finite state automata with translucent letters [13]. Here, we also consider *input-driven pushdown automata with translucent letters working in the non-returning mode* (nrNIDPDAwtl, nrDIDPDAwtl).

Let $M = \langle Q,\Sigma,\Gamma,q_0,\lhd,\bot,\tau,\delta_D,\delta_R,\delta_N\rangle$, be an nrNIDPDAwtl. Now, a configuration of M is a pair $(uqw\lhd,\gamma)$ or accept, where $q\in Q$ is the current state, $uw\in\Sigma^*$ is the remaining part of the input with w being to the right and u to the left of the input head, and $\gamma\in\Gamma^*\bot$ is the current pushdown content. The successor configuration yielded by $\vdash$ is now specified as follows. Let Let $q,q'\in Q$, $x\in\tau(q)^*$, $a\notin\tau(q)$, $u,y\in\Sigma^*$, and $z\in\Gamma$, $\gamma\in\Gamma^*\bot$. Then:

1. $(uqxay\lhd,z\gamma)\vdash(uxq'y\lhd,z'z\gamma)$, if $a\in\Sigma_D$ and $(q',z')\in\delta_D(q,a,z)$,
2. $(uqxay\lhd,\bot)\vdash(uxq'y\lhd,z'\bot)$, if $a\in\Sigma_D$ and $(q',z')\in\delta_D(q,a,\bot)$,
3. $(uqxay\lhd,z\gamma)\vdash(uxq'y\lhd,\gamma)$, if $a\in\Sigma_R$ and $q'\in\delta_R(q,a,z)$,
4. $(uqxay\lhd,\bot)\vdash(uxq'y\lhd,\bot)$, if $a\in\Sigma_R$ and $q'\in\delta_R(q,a,\bot)$,
5. $(uqxay\lhd,z\gamma)\vdash(uxq'y\lhd,z\gamma)$, if $a\in\Sigma_N$ and $q'\in\delta_N(q,a,z)$,
6. $(uqxay\lhd,\bot)\vdash(uxq'y\lhd,\bot)$, if $a\in\Sigma_N$ and $q'\in\delta_N(q,a,\bot)$,
7. $(uqx\lhd,z\gamma)\vdash(q'ux\lhd,z\gamma)$, if $q'\in\delta_N(q,\lhd,z)$,
8. $(uqx\lhd,\bot)\vdash(q'ux\lhd,\bot)$, if $q'\in\delta_N(q,\lhd,\bot)$.

In addition, whenever, the transition function yields accept for the current configuration, the successor configuration is accept.

The accepted language $L(M)$ can be easily defined. Sometimes, we will be saying that an nrNIDPDAwtl performs *sweeps*, where a sweep is a sequence of transitions that starts with the input head at the left end of the (remaining) input and ends after the next (if any) return move on the endmarker. Let us give an intuition on how an nrNIDPDAwtl works by the following example.

Example 2. We consider two languages over the alphabet $\{a,b\}$ together with its overlined variant $\{\bar{a},\bar{b}\}$ and its doubly-overlined variant $\{\bar{\bar{a}},\bar{\bar{b}}\}$. In addition, we consider two mappings h_1 and h_2 such that $h_1(a)=\bar{a}$, $h_1(b)=\bar{b}$, $h_2(a)=\bar{\bar{a}}$, and $h_2(b)=\bar{\bar{b}}$. Then, the languages L_1 and L_2 are defined as follows.

$$
\begin{aligned}
L_1 &= \{\,wvh_2(w^R)\mid w\in\{a,b\}^+, v\in\{\bar{a},\bar{b}\}^+\,\},\\
L_2 &= \{\,wh_1(w^R)v\mid w\in\{a,b\}^+, v\in\{\bar{\bar{a}},\bar{\bar{b}}\}^+\}.
\end{aligned}
$$

We will show that the union $L = L_1 \cup L_2$ is accepted by a nondeterministic input-driven pushdown automaton with translucent input letters in the non-returning mode. We define an nrNIDPDAwtl $M = \langle Q, \Sigma, \Gamma, q_0, \lhd, \bot, \tau, \delta_D, \delta_R, \delta_N \rangle$ accepting L as follows.

- $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$,
- $\Sigma_D = \{a, b\}$, $\Sigma_R = \{\bar{a}, \bar{b}, \bar{\bar{a}}, \bar{\bar{b}}\}$, $\Sigma_N = \emptyset$,
- $\Gamma = \{A, B\}$,
- $\tau(q_0) = \tau(q_2) = \tau(q_3) = \tau(q_4) = \tau(q_5) = \emptyset, \quad \tau(q_1) = \{\bar{a}, \bar{b}\}$.

In a first phase, M reads input symbols a or b and pushes corresponding symbol A or B onto the pushdown store. At any time step, M decides nondeterministically whether it remains in this phase by remaining in state q_0 or whether it starts to test whether the input belongs to the first set of the union L_1 by entering state q_1 or to the second set L_2 by entering state q_3.

To test whether the input belongs to L_1, M enters state q_1 and all symbols $\bar{a}$ or $\bar{b}$ become translucent. Then, M starts to match all remaining doubly-overlined input symbols against the pushdown store. To ensure the correct format of the remaining input, M changes its state whenever the first doubly-overlined symbol is read. If M reaches the endmarker and the pushdown store is empty, the accepting state is entered. This is realized in rules (3)–(7).

To test whether the input belongs to L_2, M enters state q_3 and no symbols are translucent. Then, M starts to match all following single-overlined input symbols against the pushdown store. Once all such symbols are read and the pushdown store is empty, M proceeds to read the doubly-overlined symbols while popping from the (already empty) pushdown store. To ensure the correct format of the input, M changes its state whenever the first overlined and the first doubly-overlined symbol is read. If M encounters the endmarker and the pushdown store is empty, M enters the accepting state. This is realized in rules (8)–(16).

The transition functions are defined as follows for $Z \in \{\bot, A, B\}$.

(1) $\delta_D(q_0, a, Z) = \{(q_0, A), (q_1, A), (q_3, A)\}$,
(2) $\delta_D(q_0, b, Z) = \{(q_0, B), (q_1, B), (q_3, B)\}$,
(3) $\delta_R(q_1, \bar{\bar{a}}, A) = \{q_2\}$,
(4) $\delta_R(q_1, \bar{\bar{b}}, B) = \{q_2\}$,
(5) $\delta_R(q_2, \bar{\bar{a}}, A) = \{q_2\}$,
(6) $\delta_R(q_2, \bar{\bar{b}}, B) = \{q_2\}$,
(7) $\delta_N(q_2, \lhd, \bot) = \texttt{accept}$,
(8) $\delta_R(q_3, \bar{a}, A) = \{q_4\}$,
(9) $\delta_R(q_3, \bar{b}, B) = \{q_4\}$,
(10) $\delta_R(q_4, \bar{a}, A) = \{q_4\}$,
(11) $\delta_R(q_4, \bar{b}, B) = \{q_4\}$,
(12) $\delta_R(q_4, \bar{\bar{a}}, \bot) = \{q_5\}$,
(13) $\delta_R(q_4, \bar{\bar{b}}, \bot) = \{q_5\}$,
(14) $\delta_R(q_5, \bar{\bar{a}}, \bot) = \{q_5\}$,
(15) $\delta_R(q_5, \bar{\bar{b}}, \bot) = \{q_5\}$,
(16) $\delta_N(q_5, \lhd, \bot) = \texttt{accept}$.

Clearly, if there is any error in the format of the input or the part of input compared to the pushdown store does not match, the transition function of M is not defined and, therefore, the input is rejected.

We would like to note that M uses its translucency to "overlook" the overlined part of an input belonging to L_1, since otherwise the pushdown store may be inadvertently emptied and could not be matched with the doubly-overlined input w^R. Moreover, M uses the non-returning mode to ensure that the input is correctly formatted, that is, symbols from $\{a, b\}^+$ are followed by symbols from $\{\bar{a}, \bar{b}\}^+$ which are in turn followed by symbols from $\{\bar{\bar{a}}, \bar{\bar{b}}\}^+$. ■

As is known for finite automata and regular pushdown automata [9], also for input-driven pushdown automata with translucent letters it holds that any automaton M working in the returning mode can be simulated by some automaton M' working in the non-returning mode, where both share the same

signature and the same transclucency mapping (for the states of M). The construction of M' for a given M, roughly speaking, works as follows: at each step, M' simulates the step of M, followed by a new step which brings the input head to the leftmost input symbol. To this end, let Q' be a primed copy of Q. The transition function δ_i, for $i \in \{D,R,N\}$, is modified to δ'_i so that the state $q' \in Q'$ is entered if and only if M enters the state $q \in Q$. The translucency mapping τ is extended to τ' by adding $\tau'(q') = \Sigma$, for all $q' \in Q'$. This clearly implies that, whenever in any state from Q', M' sees the endmarker.

Finally, δ'_N is extended by $\delta'_N(q', \lhd, z) = \{q\}$, for all $q' \in Q'$ and all $z \in \Gamma \cup \{\bot\}$, thus bringing the input head to the leftmost position. One may easily verify that M' accepts if and only M accepts.

3 Determinism versus Nondeterminism

It is well-known that for input-driven pushdown automata, deterministic devices are as powerful as their nondeterministic counterparts. This contrasts with the general case of pushdown automata, where the deterministic variant is strictly weaker than the nondeterministic one. In the following, we examine the situation for deterministic and nondeterministic input-driven pushdown automata with translucent input letters. It turns out that the family of languages accepted by deterministic input-driven pushdown automata with translucent input letters is a proper subset of the family of languages accepted by their nondeterministic counterparts, both in the returning and in the non-returning case.

Theorem 3. *The family $\mathscr{L}(\mathsf{DIDPDAwtl})$ is properly included in the family $\mathscr{L}(\mathsf{NIDPDAwtl})$ and the family $\mathscr{L}(\mathsf{nrDIDPDAwtl})$ is properly included in the family $\mathscr{L}(\mathsf{nrNIDPDAwtl})$.*

Proof. We use the union $L = L_1 \cup L_2$ as witness language for the properness of the inclusions, where $L_1 = \{\, b^n\#b^m \sqcup\!\sqcup\, a^n \mid m,n \geq 1 \,\}$ and $L_2 = \{\, b^m\#b^n \sqcup\!\sqcup\, a^n \mid m,n \geq 1 \,\}$.

Similarly as in Example 1, two DIDPDAwtl's can be constructed that accept L_1 respectively L_2, and an NIDPDAwtl can be constructed that accepts $L = L_1 \cup L_2$.

It remains to be shown that L is not accepted by any nrDIDPDAwtl. Assume in contrast to the assertion that L is accepted by some nrDIDPDAwtl $M = \langle Q, \Sigma, \Gamma, q_0, \lhd, \bot, \tau, \delta_D, \delta_R, \delta_N \rangle$. We consider accepting computations on inputs of the form $a^n b^k \# b^\ell$ where n,k,ℓ are large enough.

A basic observation is that M cannot access the b-block following the # unless the # is read or there are no more b's in front of the #.

First we claim that M cannot read the symbol # and return to the left of the input before one or both b-blocks are read entirely.

Assume in contrast to the claim that M is in some configuration $(a^{m_1} q_1 b^{k_1} \# b^{\ell_1} \lhd, \gamma_1)$ such that $b \in \tau(q_1)$, and from the successor configuration $(a^{m_1} b^{k_1} q_2 b^{\ell_1} \lhd, \gamma_2)$ it reads some further b's and then jumps to the endmarker reaching a configuration $(q_3 a^{m_1} b^{k_1} b^{\ell_2} \lhd, \gamma_3)$. If neither b^{k_1} nor b^{ℓ_2} is empty then also the inputs $a^n b^{k+1} \# b^{\ell-1}$ and $a^n b^{k-1} \# b^{\ell+1}$ would be accepted but at least one of them does not belong to L. The contradiction shows the claim.

Next, we have to consider four cases.

Case 1: The first case is that neither $a \in \Sigma_D$ and $b \in \Sigma_R$ nor $a \in \Sigma_R$ and $b \in \Sigma_D$. Essentially, this means that M does not use its pushdown.

A single sweep of M is analyzed. If M reads more than $2|Q|$ consecutive symbols a (respectively b), it must eventually enter a loop of, say c, states. Since the pushdown is not used, we can increase the length of the a-block (respectively b-block) by c without changing the overall result of the computation, a contradiction. Hence, in this case, in any sweep M cannot enter a loop while reading a's or b's. Therefore, M must perform multiple sweeps without loops. For the states in which the sweeps start there

are at most $|Q|$ possibilities. Due to the deterministic behavior, eventually M will run through loops of sweeps with respect to the starting state of the loop. Let x_0 be the number of a's read before this sweep loop and let x_1 be the number of a's consumed in one sweep loop. Similarly, let y_0 and y_1 be the numbers of b's consumed. Now we consider the input $a^{x_0+x_1}b^{y_0+y_1}\#b^\ell$ where $\ell > 2|Q|$. Then, after the first sweep loop, the a-block and the first b-block are entirely read. If $x_0+x_1 \neq y_0+y_1$, M must verify whether $x_0+x_1=\ell$ To do so, M needs to read the second block of b's – but without using the pushdown store and without any leftover a's. Given that $\ell > 2|Q|$, M must enter a state loop while reading the second b-block, say of length c'. Hence, if M accepts the input for some ℓ, it accepts the input with $\ell + c'$, which contradicts the definition of L. If $x_0+x_1=y_0+y_1$, almost the same argument holds for the input $a^{x_0+x_1}b^{y_0+y_1-1}\#b^\ell$.

Case 2: The second case is that $a \in \Sigma_D$ and $b \in \Sigma_R$. Moreover, M does not enter loops with respect to the current state and the topmost pushdown symbol.

In this case, in one sweep, M reads at most $|Q|(|\Gamma|+1)$ symbols of the a-block and the first b-block (as long as these blocks are non-empty). We argue similarly as in Case 1. Due to the deterministic behavior, eventually M will run through loops of sweeps with respect to the starting state of the loop. Let x_0 be the number of a's read before this sweep loop and let x_1 be the number of a's consumed in one sweep loop. Similarly, let y_0 and y_1 be the numbers of b's consumed.

If $x_0+x_1 > y_0+y_1$, we consider the input $a^{x_0+x_1}b^{x_0+x_1+y_0+y_1}\#b^\ell$ where $\ell > 2|Q|$. Then, after the first sweep loop, all a's are read, and the remaining input is $b^{x_0+x_1}\#b^\ell$. Moreover, there are $x_0+x_1-y_0-y_1$ symbols left in the pushdown store. If $x_0+x_1 > y_0+y_1$, next M will empty its pushdown store after reading $x_0+x_1-y_0-y_1$ many b's. Afterward, M can only rely on its finite set of states. Hence, after reading at most $|Q|$ additional symbols b, M enters a state loop of some length c''. Consequently, if M accepts the input for some ℓ it must also accept the input with $\ell + c''$, which contradicts the definition of L.

The same reasoning applies for the input $a^{x_0+x_1}b^{y_0+y_1}\#b^\ell$, where $\ell > 2|Q|$, if $x_0+x_1 < y_0+y_1$, and for the input $a^{x_0+x_1}b^{y_0+y_1-1}\#b^\ell$, where $\ell > 2|Q|$, if $x_0+x_1=y_0+y_1$.

Case 3: The third case is that $a \in \Sigma_D$ and $b \in \Sigma_R$ and M enters a loop with respect to the current state and the topmost pushdown symbol while reading an a-block or a b-block.

First, consider the a-block. After reading the a-block, M has n symbols stored in the pushdown. In a sweep where the # symbol is read, M must either read the first b-block or the second b-block entirely. Assume it reads the first b-block and consider the input $a^nb^k\#b^\ell$ with $k > n$. Then, after reading this block, the pushdown store is empty, and the rest of the input must be processed using only the finite control. As a result, the length of the second b-block can be increased which is again a contradiction. The same argument applies analogously if the second b-block is read first.

Second, consider one of the b-blocks. Since $b \in \Sigma_R$, M runs through a state loop with empty pushdown while processing the block. So the length of the block can be increased by the length of the loop without changing the overall result of the computation, a contradiction.

Case 4: The fourth case is that $a \in \Sigma_R$ and $b \in \Sigma_D$ and M enters a loop with respect to the current state and the topmost pushdown symbol while reading an a-block or a b-block. The argumentation in this case is symmetric to Case 3. Here the roles of a and b (that is, their memberships in Σ_D and Σ_R) are switched.

Finally, from the contradictions in all possible cases we conclude that L is not accepted by any nrDIDPDAwtl. □

The additional power of nondeterministic input-driven pushdown automata with translucent input letters versus their deterministic counterpart is due to the fact that the translucency of input letters al-

lows different computation paths to treat the same input symbol differently, thereby enabling different operations on the pushdown store. In a nondeterministic setting, this flexibility permits branching into multiple computational paths, each potentially using the pushdown store in a distinct way. However, a deterministic machine cannot simulate these differing operations simultaneously, which limits its expressive power.

4 Returning versus Non-Returning

It is known that deterministic pushdown automata with translucent letters working in the non-returning mode can accept even a non-semilinear language [9]. So a natural question is whether the same still holds for the structurally weaker device which has to obey the input-driven mode. The next theorem answers this question in the affirmative. To this end, we tweak the witness language from [9]. We define the non-semilinear language L_{nsl} as

$$L_{nsl} = \{\, a\,\$\#\,a^3\,\$\#^2\,a^5\,\$\#^3 \cdots \$\#^{k-1}\,a^{2k-1}\,\$\mathrm{¢}^k\,a^{2k+1} \mid k \geq 0 \,\}.$$

Theorem 4. *The language L_{nsl} is accepted by some nrDIDPDAwtl.*

The language L_{nsl} is accepted by some nrDIDPDAwtl and thus by some nrNIDPDAwtl. It is known that all languages accepted by DPDAwtl are semilinear [9]. So, the next question is whether this is still true when we trade nondeterminism for the input-driven property. The following result has been shown in [15] for finite automata with translucent letters and can be adapted for our purposes.

Proposition 5. *From any given NIDPDAwtl M, an NPDA M' can effectively be constructed, such that $L(M') \subseteq L(M)$ and $L(M')$ is letter-equivalent to $L(M)$.*

Proposition 5 together with the well-known result that all context-free languages are semilinear [24] implies that all languages in $\mathscr{L}(\mathsf{NIDPDAwtl}) \supset \mathscr{L}(\mathsf{DIDPDAwtl})$ are semilinear. So, by Theorem 4 we have the following corollary.

Corollary 6. *The family $\mathscr{L}(\mathsf{DIDPDAwtl})$ is properly included in $\mathscr{L}(\mathsf{nrDIDPDAwtl})$, and the family $\mathscr{L}(\mathsf{NIDPDAwtl})$ is properly included in $\mathscr{L}(\mathsf{nrNIDPDAwtl})$.*

The remaining two language families under consideration to be compared are $\mathscr{L}(\mathsf{nrDIDPDAwtl})$ and $\mathscr{L}(\mathsf{NIDPDAwtl})$.

Proposition 7. *The language families $\mathscr{L}(\mathsf{nrDIDPDAwtl})$ and $\mathscr{L}(\mathsf{NIDPDAwtl})$ are incomparable.*

Proof. Theorem 4 provides a non-semilinear language accepted by some nrDIDPDAwtl but not accepted by any NIDPDAwtl.

Conversely, the proof of Theorem 3 provides a language that is accepted by some NIDPDAwtl but cannot be accepted by any nrDIDPDAwtl. □

5 Closure under Boolean Operations

Often nondeterministic devices induce language families that are closed under union but are not closed under intersection. This implies immediately the non-closure under complementation. However, here the closure under union is an open problem. Therefore, we show the non-closure under complementation directly. A witness for the nondeterministic families $\mathscr{L}(\mathsf{nrNIDPDAwtl})$ and $\mathscr{L}(\mathsf{NIDPDAwtl})$ is the language

$$L_{rep} = \{\, b^n(\#b^n)^k \sqcup\!\sqcup\, a^n \mid n \geq 1, k \geq 0 \,\}.$$

Theorem 8. *The language families $\mathscr{L}(\mathsf{nrNIDPDAwtl})$ and $\mathscr{L}(\mathsf{NIDPDAwtl})$ are not closed under complementation.*

It is well known that the families of languages induced by deterministic pushdown automata and real-time deterministic pushdown automata are closed under complementation. The closure has also been derived for DPDAwtl and nrDPDAwtl [9]. Here we complement these results by showing the closure for the deterministic language families studied here.

Proposition 9. *The language families $\mathscr{L}(\mathsf{nrDIDPDAwtl})$ and $\mathscr{L}(\mathsf{DIDPDAwtl})$ are closed under complementation.*

Next, we show that both deterministic families are not closed under the remaining Boolean operations union and intersection.

Proposition 10. *The language families $\mathscr{L}(\mathsf{nrDIDPDAwtl})$ and $\mathscr{L}(\mathsf{DIDPDAwtl})$ are neither closed under union nor under intersection.*

Proof. We use the languages $L_1 = \{ b^n\#b^m \sqcup\!\sqcup a^n \mid m,n \geq 1 \}$ and $L_2 = \{ b^m\#b^n \sqcup\!\sqcup a^n \mid m,n \geq 1 \}$ introduced in the proof of Theorem 3. Each language can be accepted by a DIDPDAwtl with signature $\Sigma_D = \{a\}$, $\Sigma_R = \{b\}$, and $\Sigma_N = \{\#\}$. The rough idea for a DIDPDAwtl M_1 accepting L_1 is to push in a first phase all a's while symbols b and # are translucent. In a second phase, the first block of b's is matched against the pushdown store and the second block of b's is in fact ignored since M_1 may pop from the empty pushdown only.

The rough idea for a DIDPDAwtl M_2 accepting L_2 is to consume all b's up to and including the symbol # in a first phase. At the end of this phase the pushdown store is empty. In a second phase, all symbols b are translucent and the a's are consumed and pushed. Finally, in a third phase, the remaining b's from the input are matched against the pushdown store.

Since it is shown in Theorem 3 that the union $L_1 \cup L_2$ is not accepted by any nrDIDPDAwtl, we obtain the non-closure under union for $\mathscr{L}(\mathsf{DIDPDAwtl})$ as well as for $\mathscr{L}(\mathsf{nrDIDPDAwtl})$, even if the given automata have identical signatures. Since both families are closed under complementation by Proposition 9, we also obtain the non-closure under intersection for $\mathscr{L}(\mathsf{DIDPDAwtl})$ as well as for $\mathscr{L}(\mathsf{nrDIDPDAwtl})$. □

For the nondeterministic families we already know the non-closure under complementation. We now show that the nondeterministic returning class is not closed under intersection even with regular languages. To this end, we use the result of Proposition 5 stating that from any given NIDPDAwtl M we can effectively construct an NPDA M' such that $L(M') \subseteq L(M)$ and $L(M')$ is letter-equivalent to $L(M)$. Since the context-sensitive language $L_{abc} = \{ a^n b^n c^n \mid n \geq 0 \}$ does not contain any letter-equivalent context-free sub-language, we can conclude that $L_{abc} \notin \mathscr{L}(\mathsf{NIDPDAwtl})$. On the other hand, $L_{abc} = L_{a,b,c} \cap a^*b^*c^*$ with $L_{a,b,c} = \{ w \in \{a,b,c\}^* \mid |w|_a = |w|_b = |w|_c \}$ and $L_{a,b,c}$ as well as the regular language $a^*b^*c^*$ can be accepted by NIDPDAwtl having identical signatures. Hence, we obtain the following proposition.

Proposition 11. *The language family $\mathscr{L}(\mathsf{NIDPDAwtl})$ is neither closed under intersection nor under intersection with regular languages.*

It remains an open question whether or not the family $\mathscr{L}(\mathsf{NIDPDAwtl})$ is closed under union and whether or not the family $\mathscr{L}(\mathsf{nrNIDPDAwtl})$ is closed under union or intersection. Obviously, we obtain the closure under union for both families if the signatures are compatible. However, we strongly conjecture non-closure in all other cases.

6 Decidability Questions

In this section, we investigate decidability questions such as, for example, emptiness, finiteness, universality, inclusion, equivalence, and regularity for our introduced input-driven variants of pushdown automata with translucent letters. These decidability questions have already been investigated for deterministic and nondeterministic finite automata with translucent letters in [16, 18] where some partial results have been obtained. For returning deterministic and nondeterministic finite automata with translucent letters the questions of emptiness and finiteness are decidable. In addition, universality is decidable in the deterministic case. Inclusion is already undecidable for returning deterministic finite automata with translucent letters. This negative result carries over to all other models in the returning and/or nondeterministic case. For returning nondeterministic finite automata with translucent letters the questions of equivalence and regularity are undecidable and carry over to non-returning nondeterministic finite automata with translucent letters as well. Here, we study these questions for pushdown automata with translucent letters. We note that some of the undecidability results obtained for finite automata with translucent letters carry over to pushdown automata with translucent letters. However, we show here the non-semidecidability of the problems and, in addition, the non-semidecidability of universality for nondeterministic input-driven pushdown automata in the returning and non-returning case. We start with decidable questions and show that some questions are decidable for returning pushdown automata with translucent letters.

Theorem 12. *For DIDPDAwtl or DPDAwtl as input, the problems of testing emptiness, finiteness, and universality are decidable. For NIDPDAwtl as input, the problems of testing emptiness and finiteness are decidable.*

Next, we switch to undecidability results and we will show, in particular, the non-semidecidability of some problems. To prove these results we use the technique of valid computations of Turing machines [5]. It suffices to consider deterministic Turing machines with one single read-write head and one single tape whose space is fixed by the length of the input, that is, so-called linear bounded automata (LBA). Without loss of generality and for technical reasons, we assume that the LBAs can halt only after an odd number of moves, accept by halting, and make at least three moves. A valid computation is a string built from a sequence of configurations passed through during an accepting computation.

Let Q be the state set of some LBA M, where q_0 is the initial state, $T \cap Q = \emptyset$ is the tape alphabet containing the endmarkers $\trianglerighteq$ and $\trianglelefteq$, and $\Sigma \subset T$ is the input alphabet. A configuration of M can be written as a string of the form $\trianglerighteq T^*QT^*\trianglelefteq$ such that, $\trianglerighteq t_1t_2\cdots t_iqt_{i+1}\cdots t_n\trianglelefteq$ is used to express that $\trianglerighteq t_1t_2\cdots t_n\trianglelefteq$ is the tape inscription, M is in state q, and is scanning tape symbol t_{i+1}.

The set of valid computations $\mathrm{VALC}(M)$ is now defined to be the set of words having the form $w_0\#w_2\#\cdots\#w_{2m}cw_{2m+1}^R\#w_{2m-1}^R\#\cdots\#w_3^R\#w_1^R$, where $\#, c \notin T \cup Q$, $w_i \in \trianglerighteq T^*QT^*\trianglelefteq$ are configurations of M, w_0 is an initial configuration from $\trianglerighteq q_0\Sigma^*\trianglelefteq$, w_{2m+1} is a halting, that is, an accepting configuration, and w_{i+1} is the successor configuration of w_i for $0 \leq i \leq 2m$. The set of invalid computations $\mathrm{INVALC}(M)$ is the defined as the complement of $\mathrm{VALC}(M)$ with respect to the alphabet $T \cup Q \cup \{\#, c\}$.

To accept the set $\mathrm{INVALC}(M)$ by an NIDPDAwtl we make some modifications. Let h' be a mapping that maps every symbol from $T \cup Q \cup \{\#\}$ to its primed version. Similarly, let h'' be a mapping that maps every symbol from $T \cup Q \cup \{\#\}$ to its double-primed version. We then define the set of valid computations $\mathrm{VALC}'(M)$ to be the set of words of the form $w_0\#w_2\#\cdots\#w_{2m}c\left(h'(w_{2m+1}^R\#) \sqcup\!\sqcup h''(w_{2m-1}^R\#\cdots\#w_3^R\#w_1^R)\right)$, where $w_0\#w_2\#\cdots\#w_{2m}cw_{2m+1}^R\#w_{2m-1}^R\#\cdots\#w_3^R\#w_1^R$ belongs to $\mathrm{VALC}(M)$. The set of invalid computations $\mathrm{INVALC}'(M)$ is then defined as the complement of $\mathrm{VALC}'(M)$.

Lemma 13. *Let M be an LBA. Then, an NIDPDAwtl accepting $\mathrm{INVALC}'(M)$ can effectively be constructed.*

Proof. We sketch the construction of an NIDPDAwtl M' accepting INVALC$'(M)$ whose signature is defined as $\Sigma_D = T \cup Q \cup \{\#\}$, $\Sigma_N = \{c\}$, and $\Sigma_R = T' \cup T'' \cup Q' \cup Q'' \cup \{\#', \#''\}$. To check whether an input x belongs to INVALC$'(M)$, M' guesses and tests one of the following four possibilities.

1. x has the wrong format to belong to VALC$'(M)$.
2. x has the correct format, but the number of configurations to the left of c is different from the number of configurations to the right of c.
3. $x = w_0\#w_2\#\cdots\#w_{2m}c\left(h'(w_{2m+1}^R\#) \sqcup h''(w_{2m-1}^R\#\cdots\#w_3^R\#w_1^R)\right)$, but w_{2i+1} is not the successor configuration of w_{2i} for some $0 \leq i \leq m$.
4. $x = w_0\#w_2\#\cdots\#w_{2m}c\left(h'(w_{2m+1}^R\#) \sqcup h''(w_{2m-1}^R\#\cdots\#w_3^R\#w_1^R)\right)$, but w_{2i+2} is not the successor configuration of w_{2i+1} for some $0 \leq i \leq m-1$.

The first possibility can be tested by a finite automaton and, hence, by M' disregarding the actions on the pushdown store. For the second possibility M' acts as follows: it reads the input up to the middle marker c and pushes the input as it is on the pushdown store. After reading the marker c, M' makes all symbols from $T'' \cup Q'' \cup \{\#''\}$ translucent and pops for every input symbol from $T' \cup Q' \cup \{\#'\}$ a symbol from the pushdown store taking care that $\#'$ in the input is only matched against $\#$ on the pushdown store. If an error is encountered in this phase, the input is accepted. If M' sees the endmarker, M' reads the remaining input and pops for every input symbol from $T'' \cup Q'' \cup \{\#''\}$ a symbol from the pushdown store taking again care that $\#''$ in the input is only matched against $\#$ on the pushdown store. If an error is encountered in this phase, the input is accepted as well. If the input is read completely and the pushdown store is not empty, or the pushdown store gets empty before the input is read completely, M' accepts as well and rejects in all other cases.

To test the third possibility M' reads the input up to the middle marker c and pushes the input as it is on the pushdown store. Additionally, at some point of time M' guesses the index i. Then, M' pushes configuration w_{2i} with suitably marked symbols on the pushdown store and M' remembers the last three symbols read in its finite control until the state symbol of configuration w_{2i} is the middle one of these three. After reading the middle marker c the task is to identify configuration w_{2i+1} in the input and to check that w_{2i+1} is not the successor configuration of w_{2i}. If the suitably marked configuration on the pushdown store is the topmost one after reading c, M' makes all symbols from $T'' \cup Q'' \cup \{\#''\}$ translucent and pops for every input symbol from $T' \cup Q' \cup \{\#'\}$ a symbol from the pushdown store verifying that the current configuration is *not* the reversal of the successor configuration of the configuration stored in the

Automata Family	$\emptyset$	FIN	Σ^*	$\subseteq$	$=$	REG
DPDAwtl	✓	✓	✓	✕	?	?
DIDPDAwtl	✓	✓	✓	✕	?	?
NIDPDAwtl	✓	✓	✗	✗	✗	✗
nrDPDAwtl	?	?	?	✕	?	?
nrDIDPDAwtl	?	?	?	✕	?	?
nrNIDPDAwtl	?	?	✗	✗	✗	✗

Table 1: A summary of decidability questions for the language families discussed in this paper. The undecidable questions derived from finite automata with translucent letters are marked with '✕', whereas the non-semidecidable questions obtained in this paper are marked with '✗'.

pushdown store. Both configurations differ only locally at the state symbol. But from the information remembered in the finite control, the differences can be computed and verified. If an error is encountered, the input is accepted and otherwise rejected. If the suitably marked configuration on the pushdown store is not the topmost one after reading c, then M' makes all symbols from $T'' \cup Q'' \cup \{\#''\}$ translucent and pops for every input symbol from $T' \cup Q' \cup \{\#'\}$ a symbol from the pushdown store checking the correct length and format as in the test of the second possibility. After this phase handling inputs from $T' \cup Q' \cup \{\#'\}$, M' reads the remaining input and pops for every input symbol from $T'' \cup Q'' \cup \{\#''\}$ a symbol from the pushdown store checking again the correct length and format as in the test of the second possibility until the suitably marked symbols appear on the pushdown store. In this case, M' pops for every input symbol from $T'' \cup Q'' \cup \{\#''\}$ a symbol from the pushdown store verifying that the current configuration is *not* the reversal of the successor configuration of the configuration stored in the pushdown store. Again, this can be computed and verified due to the information remembered in the finite control, since both configurations differ only locally at the state symbol. If an error is encountered, the input is accepted and otherwise rejected.

The idea to test the fourth possibility is in a first phase identical to the third possibility: M' reads the input up to the middle marker c and pushes the input as it is on the pushdown store. Additionally, M' pushes configuration w_{2i+2} with suitably marked symbols and remembers the last three symbols read in its finite control until the state symbol of configuration w_{2i+2} is the middle one of these three. After reading the middle marker c the task is to identify configuration w_{2i+1} in the input and to check that w_{2i+1} is not the successor configuration of w_{2i+2}. To this end, M' makes all symbols from $T' \cup Q' \cup \{\#'\}$ translucent and pops for every input symbol from $T'' \cup Q'' \cup \{\#''\}$ a symbol from the pushdown store checking the correct length and format as in the test of the second possibility until the suitably marked symbols appear on the pushdown store. In this case, M' pops for every input symbol from $T'' \cup Q'' \cup \{\#''\}$ a symbol from the pushdown store verifying that the reversal of the successor configuration of the current configuration is *not* the configuration stored in the pushdown store. Again, this can be computed and verified due to the information remembered in the finite control, since both configurations differ only locally at the state symbol. If an error is encountered, the input is accepted and otherwise rejected. We note that it is implicitly detected by possibilities 3 and 4 if all configurations do not have the same length. This completes the construction of the NIDPDAwtl M' accepting $\mathrm{INVALC}'(M)$. □

The fact that NIDPDAwtl accept the set of invalid computations of an LBA is sufficient to obtain the next non-semidecidability results.

Theorem 14. *For NIDPDAwtl or nrNIDPDAwtl as input, the problems of testing universality, inclusion, equivalence, and regularity are not semidecidable.*

References

[1] Simon Beier & Markus Holzer (2022): *Nondeterministic right one-way jumping finite automata*. Inform. Comput. 284, p. 104687, doi:10.1016/J.IC.2021.104687.

[2] Suna Bensch, Henning Bordihn, Markus Holzer & Martin Kutrib (2009): *On input-revolving deterministic and nondeterministic finite automata*. Inform. Comput. 207, pp. 1140–1155, doi:10.1016/j.ic.2009.03.002.

[3] Burchard von Braunmühl & Rutger Verbeek (1985): *Input-Driven Languages are Recognized in* $\log n$ *Space*. In Marek Karpinski & Jan van Leeuwen, editors: Topics in the Theory of Computation, Mathematics Studies 102, North-Holland, Amsterdam, pp. 1–19, doi:10.1016/S0304-0208(08)73072-X.

[4] Hiroyuki Chigahara, Szilárd Zsolt Fazekas & Akihiro Yamamura (2016): *One-Way Jumping Finite Automata*. Int. J. Found. Comput. Sci. 27, pp. 391–405, doi:10.1142/S0129054116400165.

[5] Juris Hartmanis (1967): *Context-free languages and Turing machine computations*. Proc. Symposia in Applied Mathematics 19, pp. 42–51, doi:10.1090/psapm/019/0235938.

[6] John E. Hopcroft & Jeffrey D. Ullman (1979): *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Massachusetts.

[7] Petr Jančar, František Mráz, Martin Plátek & Jörg Vogel (1995): *Restarting automata*. In Horst Reichel, editor: *Fundamentals of Computation Theory (FCT 1995)*, *LNCS* 965, Springer, pp. 283–292, doi:10.1007/3-540-60249-6_60.

[8] Martin Kutrib, Andreas Malcher, Carlo Mereghetti & Beatrice Palano (2025): *Two-Way Finite Automata with Translucent Input Letters*. In Andreas Malcher & Luca Prigioniero, editors: *Descriptional Complexity of Formal Systems (DCFS 2025)*, *LNCS* 15759, Springer, pp. 151–165, doi:10.1007/978-3-031-97100-6_11.

[9] Martin Kutrib, Andreas Malcher, Carlo Mereghetti, Beatrice Palano, Priscilla Raucci & Matthias Wendlandt (2024): *Deterministic Pushdown Automata with Translucent Input Letters*. In Joel D. Day & Florin Manea, editors: *Developments in Language Theory (DLT 2024)*, *LNCS* 14791, Springer, pp. 203–217, doi:10.1007/978-3-031-66159-4_15.

[10] Martin Kutrib, Andreas Malcher, Carlo Mereghetti, Beatrice Palano, Priscilla Raucci & Matthias Wendlandt (2024): *On Properties of Languages Accepted by Deterministic Pushdown Automata with Translucent Input Letters*. In Szilárd Zsolt Fazekas, editor: *Implementation and Application of Automata (CIAA 2024)*, *LNCS* 15015, Springer, pp. 208–220, doi:10.1007/978-3-031-71112-1_15.

[11] Alexander Meduna & Petr Zemek (2012): *Jumping finite automata*. Int. J. Found. Comput. Sci. 23, pp. 1555–1578, doi:10.1142/S0129054112500244.

[12] Kurt Mehlhorn (1980): *Pebbling Mountain Ranges and its Application of DCFL-Recognition*. In J. W. de Bakker & Jan van Leeuwen, editors: *International Colloquium on Automata, Languages and Programming (ICALP 1980)*, *LNCS* 85, Springer, pp. 422–435, doi:10.1007/3-540-10003-2_89.

[13] Frantisek Mráz & Friedrich Otto (2023): *Non-returning deterministic and nondeterministic finite automata with translucent letters*. RAIRO Theor. Informatics Appl. 57, p. 8, doi:10.1051/ITA/2023009.

[14] Benedek Nagy & Friedrich Otto (2011): *CD-systems of stateless deterministic R(1)-automata governed by an external pushdown store*. RAIRO Theor. Informatics Appl. 45, pp. 413–448, doi:10.1051/ITA/2011123.

[15] Benedek Nagy & Friedrich Otto (2011): *Finite-state Acceptors with Translucent Letters*. In G. Bel-Enguix, V. Dahl & A.O. De La Puente, editors: *International Workshop on AI Methods for Interdisciplinary Research in Language and Biology (BILC 2011)*, SciTePress, pp. 3–13, doi:10.5220/0003272500030013.

[16] Benedek Nagy & Friedrich Otto (2012): *On CD-systems of stateless deterministic R-automata with window size one*. J. Comput. Syst. Sci. 78, pp. 780–806, doi:10.1016/J.JCSS.2011.12.009.

[17] Benedek Nagy & Friedrich Otto (2013): *Deterministic pushdown-CD-systems of stateless deterministic R(1)-automata*. Acta Inform. 50, pp. 229–255, doi:10.1007/S00236-012-0175-X.

[18] Benedek Nagy & Friedrich Otto (2013): *Globally deterministic CD-systems of stateless R-automata with window size 1*. Int. J. Comput. Math. 90(6), pp. 1254–1277, doi:10.1080/00207160.2012.688820.

[19] Benedek Nagy & Friedrich Otto (2024): *Finite Automata with Sets of Translucent Words*. In Joel D. Day & Florin Manea, editors: *Developments in Language Theory (DLT 2024)*, *LNCS* 14791, Springer, pp. 236–251, doi:10.1007/978-3-031-66159-4_17.

[20] Alexander Okhotin & Kai Salomaa (2014): *Complexity of input-driven pushdown automata*. SIGACT News 45, pp. 47–67, doi:10.1145/2636805.2636821.

[21] Friedrich Otto (2015): *On Visibly Pushdown Trace Languages*. In Giuseppe F. Italiano, Tiziana Margaria-Steffen, Jaroslav Pokorný, Jean-Jacques Quisquater & Roger Wattenhofer, editors: *SOFSEM 2015*, *LNCS* 8939, Springer, pp. 389–400, doi:10.1007/978-3-662-46078-8_32.

[22] Friedrich Otto (2023): *A Survey on automata with translucent letters*. In Benedek Nagy, editor: *Implementation and Application of Automata (CIAA 2023)*, *LNCS* 14151, Springer, pp. 21–50, doi:10.1007/978-3-031-40247-0_2.

[23] Friedrich Otto (2025): *Restarting Automata*. Springer, Cham, Switzerland, doi:10.1007/978-3-031-78701-0.

[24] Rohit J. Parikh (1966): *On Context-Free Languages*. *J. ACM* 13, pp. 570–581, doi:10.1145/321356.321364.

Orchestration of Music by Grammar Systems

Jozef Makiš
Faculty of Information Technology,
Brno University of Technology,
Brno, Czech republic
xmakis00@stud.fit.vut.cz

Alexander Meduna
Faculty of Information Technology,
Brno University of Technology,
Brno, Czech republic
meduna@fit.vut.cz

Zbyněk Křivka
Faculty of Information Technology,
Brno University of Technology,
Brno, Czech republic
krivka@fit.vut.cz

This application-oriented study concerns computational musicology, which makes use of grammar systems. We define multi-generative rule-synchronized scattered-context grammar systems (without erasing rules) and demonstrates how to simultaneously make the arrangement of a musical composition for performance by a whole orchestra, consisting of several instruments. Primarily, an orchestration like this is illustrated by examples in terms of classical music. In addition, the orchestration of jazz compositions is sketched as well. The study concludes its discussion by suggesting five open problem areas related to this way of orchestration.

1 Introduction

Formal languages and their models, such as automata and grammars, represent a well-developed body of knowledge, which fulfill a crucially important role in theoretical computer science as a whole. Indeed, these models, such as Turing machines, have allowed this science to establish the very fundamentals of computation, including such key areas as computability, decidability, or computational complexity. From a practical viewpoint, there also exist engineering applications of these models; for instance, compiler writing customarily makes use of finite and pushdown automata, regular expressions, and context-free grammars. Nevertheless, admittedly, the significance of these models in theory somewhat exceeds that of their use in practice. To reduce this theory-versus-practice imbalance, researchers have struggled to use and apply these models in a variety of creative areas concerning not only science but also art, such as visual art made by automata (see [1]). Recently, researchers have also studied how to use automata or grammars, such as classical generative grammars or L systems, in musicology (see [3–8, 10–12, 14–18, 22–24, 26, 30, 31]). The present paper contributes to this modern application-oriented trend concerning the use of language models to compose music.

Up until now, all the studies concerning the use of language models in music have restricted their investigation to the composition of a music score for a single instrument, such as piano. The fundamental goal of the present paper consists in a generalization of this investigation so it simultaneously produce a score for several instruments. In other words, this application-oriented study demonstrates how to make the arrangement of a musical composition for performance by a whole orchestra. Simply and plainly put, it shows how to orchestrate music based upon language models.

More specifically, consider an n-instrument orchestra, where n is a natural number; for example, for a nonet, $n = 9$. In this paper, we describe how to produce a score for this orchestra by using a grammar system consisting of n grammatical components, represented by scattered context grammars (without erasing rules) in this paper. In terms of the orchestra, every component corresponds to one of the n instruments, and its goal consists in the generation of the score for the corresponding instrument. During a generative step made by the n-component system, all the components work in parallel, and the selection of the rules applied in every single component is globally synchronized across the system as

N. Moreira and L. Prigioniero (Eds.): 15th International Workshop on
Non-Classical Models of Automata and Applications (NCMA 2025).
EPTCS 422, 2025, pp. 45–58, doi:10.4204/EPTCS.422.4

a whole. This synchronization is arranged by a finite number of prescribed n-rule sequences so that the system selects one of these sequences and applies its jth rule in the ith component, $1 \leq i \leq n$. Once a sequence of n terminal strings is generated by repeatedly making generative steps in the way sketched above, the generative process stops. From a musicological standpoint, the resulting sequence generated in this way represents the score for the whole n-instrument orchestra in such a way that the ith terminal string represents the score for the ith instrument.

The present paper is organized as follows. Section 2 recalls all the terminology needed in this paper. Section 3 defines the notion of a rule-synchronized grammar system with scattered context components. Section 4, which represents the heart of the present study, explains how to use these systems to generate multi-instrument score. Section 5 illustrates this by an example. Section 6 evaluates the proposed method in the context of music generation using formal models. Section 7 closes all the study by its summarization and a formulation of important open problem areas concerning the subject of this paper.

2 Preliminaries

We assume that the reader is familiar with discrete mathematics, and formal theory (see [2, 9]) as well as formal language theory (see [21, 28, 29]).

For a set W, card(W) denotes its *cardinality*. An *alphabet* is a finite nonempty set—elements are called *symbols*. Let V be an *alphabet*. V^* is the *set of all strings* over V. Algebraically, V^* represents the free monoid generated by V under the operation of concatenation. The identity of V^* is denoted by ε. Set $V^+ = V^* - \{\varepsilon\}$. Algebraically, V^+ is thus the free semigroup generated by V under the operation of concatenation. For $w \in V^*$, $a \in V$, and $A \subseteq V$, $|w|$ denotes the *length of* w, $\#_a(w)$ denotes the *number of occurrences of the symbol* a *in* w, and $\#_A(w)$ denotes the *number of occurrences of the symbols from* A *in* w. The *alphabet of* w, denoted by alph(w), is the set of symbols appearing in w.

Let $\Rightarrow$ be a relation over V^*. We denote ith power of $\Rightarrow$ as $\Rightarrow^i$, for $i \geq 0$. The transitive and the transitive-reflexive closure of $\Rightarrow$ are denoted by $\Rightarrow^+$ and $\Rightarrow^*$, respectively. Unless we explicitly stated otherwise, we write $x \Rightarrow y$ instead of $(x,y) \in \Rightarrow$ throughout.

3 Definitions

The present section defines the language theory notions used throughout the rest of this paper. First, it defines scattered context grammars, which represent well-known grammatical model. Then, based upon these grammars, it introduces rule-synchronized music grammar systems, which are later used as an orchestration formalism for music.

Definition 1 *A* scattered context grammar *is a quadruple, $G = (N,T,P,S)$, where N and T are alphabets such that $N \cap T = \emptyset$. Symbols in N are referred to as nonterminals while symbols in T are terminals. N contains S—the start symbol of G. P is a finite non-empty set of rules such that every $p \in P$ has the form*

$$(A_1,\ldots,A_n) \rightarrow (x_1,\ldots,x_n),$$

where $n \geq 1$, and for all $i = 1,\ldots,n$, $A_i \in N$ and $x_i \in (N \cup T)^*$. If each x_i satisfies $|x_i| \leq 1$, $i = 1, \ldots,n$, then $(A_1,\ldots,A_n) \rightarrow (x_1,\ldots,x_n)$ is said to be simple. If $n = 1$, then $(A_1) \rightarrow (x_1)$ is referred to as a context-free rule; for brevity, we hereafter write $A_1 \rightarrow x_1$ instead of $(A_1) \rightarrow (x_1)$. If for some $n \geq 1$, $(A_1, \ldots,A_n) \rightarrow (x_1,\ldots,x_n) \in P$, $v = u_1A_1u_2 \cdots u_{n-1}A_nu_n$, and $w = u_1x_1u_2 \cdots u_{n-1}x_nu$ with $u_i \in (N \cup T)^*$ for all $i = 1,\ldots,n$, then v directly derives w in G, symbolically written as $v \Rightarrow w\ [(A_1,\ldots,A_n) \rightarrow (x_1,\ldots,x_n)]$

or, simply, $v \Rightarrow w$ in G. In the standard manner, extend $\Rightarrow$ to $\Rightarrow^n$, where $n \geq 0$; then, based on $\Rightarrow^n$, define $\Rightarrow^+$ and $\Rightarrow^*$. The language of G, $L(G)$, is defined as $L(G) = \{w \in T^* \mid S \Rightarrow^* w\}$. A derivation of the form $S \Rightarrow^* w$ with $w \in T^*$ is called a successful derivation.

Next, we define the notion of an n-generative rule-synchronized music grammar system as the central notion of this paper as a whole. In essence, this notion is based upon that of an n-generative rule-synchronized music grammar system with context-free components (see [15] and Section 13.3 in [19]), but the new notion is underlain by scattered context components.

Definition 2 *An n-generative rule-synchronized music grammar system is defined as an $(m+1)$-tuple*

$$G_s = (G_1, \ldots, G_m, Q),$$

in which

- *$G_i = (N_i, T_i, P_i, S_i)$ is a scattered context grammar introduced in Definition 1, for all $i = 1, \ldots, m$;*
- *Q is a finite set that consists of n-tuples structured as $(p_1, p_2, \ldots, p_m)$, where $p_i \in P_i$, for all $i = 1, \ldots, m$.*

In addition to the original definition, we will use tokens instead of plain terminals. Tokens have indexed attributes they represent that are going to be taken into account in the final music interpretation by the instrument. Tokens are in the form $t_{[w_1,w_2,\ldots,w_k]} \in T_i$, where $w_1, w_2, \ldots, w_k$ are music attributes like tone length, special operation (tone inversion, shift, etc.), chord or others. Number k expresses the number of token attributes.

To improve readability while generating harmonic passages in music, we chose to represent chords using symbols from the Greek alphabet for simplicity, as they are difficult to denote with single-character symbols. In the example, there are mappings of symbols from Greek alphabet to chords.

The terminal strings derived from the start symbol of a grammar or in our model are in m-form as m-tuples structured as $S_f = (x_1, \ldots, x_m)$, where $x_i \in T^*$, for all $i = 1, \ldots, m$. Let us take

$$c_i = a_1 A_1 \cdots a_{n-1} A_n a_n,$$

$$d_i = a_1 x_1 \cdots a_{n-1} x_n a_n.$$

Then $S_f = (c_1, c_2, \ldots, c_m)$ and $\bar{S}_f = (d_1, d_2, \ldots, d_m)$ are sentential m-forms, in which $c_i, d_i \in (N \cup T)^*$, for every $i = 1, \ldots, m$. Consider r_i: $(A_1, \ldots, A_n) \rightarrow (x_1, \ldots, x_n) \in P_i$ for all $i = 1, \ldots, m$ and $(r_1, r_2, \ldots, r_m) \in Q$, such that $r_i = c_i \rightarrow d_i$. Consequently, S_f directly derives $\bar{S}_f$ in G_s, denoted by

$$S_f \Rightarrow_{G_s} \bar{S}_f.$$

Let us generalize $\Rightarrow_{G_s}$ with $\Rightarrow^k_{G_s}$, for all $k \geq 0$, $\Rightarrow^+_{G_s}$ and $\Rightarrow^*_{G_s}$. Generated m-string of G_s, denoted by m-$S(G_s)$, we define by

$$m\text{-S}(G_s) = \{(w_1, \ldots, w_m) \mid (S_1, \ldots, S_m) \Rightarrow^*_{G_s} (w_1, \ldots, w_m),$$

$$w_i \in T^*, \text{for all } i = 1, \ldots, m\}.$$

4 Orchestration

Building on the concepts and formalisms introduced in the previous section, this part of the work is focused on the orchestration process across multiple instruments. What led us to this is the work of others that are dealing with the algorithmic composition and grammar-based music generation. The popularity of grammar-based approaches has started with interesting applications using L-systems [24] where generated string is interpreted as a sequence of notes. This research was expanded in the works of [5, 8, 18, 27, 30] and many others. A doctoral dissertation explored automata driven by rhythm in musical improvisation [26]. It may seem like the L-systems rule the grammar-based approaches but that is just not true. The diversity of grammatical frameworks has been explored in the literature. For instance, [31] investigates hierarchical structure-building mechanisms across music, language, and animal song using formal language theory. By using context-free grammars, [14] describes how to model jazz improvisation within a controlled generative system. The notion of a probabilistic context-free grammar specifically tailored for melodic reduction is discussed in [7]. Furthermore, [11] presents a formal semantic framework to model control flow in Western music notation. Similarly, [22] applies probabilistic temporal graph grammars to model music as a language. In [6], a procedural music generation by using formal grammars is explored. Finally, [10] applies grammar-based compression techniques to uncover structural patterns in music.

While some of the cited works are capable of capturing both context-free and non-context-free dependencies (see Fig. 1), as discussed in [12], they fall short when it comes to modeling the complex interactions present in multi-instrumental compositions. By context-free and non-context-free dependencies, we refer to nested and crossing connections between notes, respectively. For this reason, we have chosen to use an n-generative rule-synchronized music grammar system, which allows the system to make the simultaneous rewriting of multiple nonterminals. This property makes them well-suited to represent interdependent musical structures that occur in music. As a basic example, we can take the piano, which can have written harmony in the bass clef and written melody in the treble clef. Or two instruments like the piano and violin may complement one another to produce a richer and more engaging melodic texture.

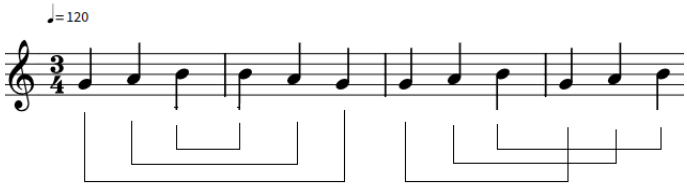

Figure 1: Context-free (1st half) and non-context free dependencies (2nd half).

As a component of our grammar system, context-free grammar would not be just enough. As a demonstration, we can take a look at Fig. 1. Starting from context-free grammars, we can describe well-connected melodies. Well-connected melodies go somewhere and return in a similar way, but such structures are not common in music. More commonly, repetition and variation create crossing dependencies, such as the ones we can see in the second half of the figure. This approach fits classical and jazz music, but it can be applied almost in any structural music.

Encoding Musical Concepts into the Grammar

To showcase our model, we have picked the sonata form from classical music, and jazz music is represented by its standard form. Mentioned forms presented here are taken from [25] and [13].

We have decided to talk about two examples to demonstrate how musical pieces could be encoded into grammar. The first is popular jazz song Take The A Train from [25]. The second is [23] and shows a minimalistic example of sonata form called Allegro in F composed by Mozart.

When choosing a top-down approach to analyze a musical piece, we start by examining its overall structure. A great example is the jazz song [25], which uses the most common structure in jazz standards, the *AABA* form. This song consists of two distinct sections (A and B), with each section typically spanning eight measures. These sections form the standard 32-measure framework of the basic melody found in *AABA* jazz compositions.

When applying a similar analytical approach to the sonata form, we observe a three-part structure: exposition (A), development (B), and recapitulation (A'). The exposition introduces the primary thematic material, typically divided into two contrasting themes. The development explores these themes through variations, modulations, and transformations. Finally, the recapitulation returns to the original thematic material, usually restating the exposition themes in their original keys or slightly modified. This structured approach allows composers to achieve a coherent and varied musical narrative, which is fundamental to classical sonata compositions.

The from can vary in different compositions, styles. For example, we can generate the *AABA* or ABA' form with following rules:

$$\begin{aligned} & S \to AABA \\ \text{or} \quad & S \to ABA'. \end{aligned}$$

Encoding Melody and Harmony

Once we have generated the initial nonterminals that outline the structure of the musical piece, the next step is to create the actual musical content. Music is truly creative, and there are endless possibilities. In our sonata example, we could encode exposition into three non-terminals T_1, R, T_2 and similarly recapitulation T_1', R', T_2'. The symbol R represents the transitions between the tonic and dominant phrases T_1 and T_2. T_1 and T_2 are also themes of our song that create interesting tension. Development in an example could be characterized by two variations of original theme and we will denote it by V_1 and V_2. To put this into rules

$$\begin{aligned} (A,A') &\to (T_1RT_2,\ T_1'R'T_2'), \\ B &\to (V_1,\ V_2). \end{aligned}$$

For our jazz example, we first introduce the main theme and then repeat it, perhaps with slight variations. These two A sections are followed by a section known as the bridge, characterized by contrasting melody or harmony. Finally, the original main theme returns. Each of these sections typically consists of eight measures. In the jazz piece we have selected we have a theme from two similar melodies. Rules that would generate structure would look like:

$$\begin{aligned} (A,A,A) &\to (T_1T_2,\ T_1T_2,\ T_1T_2), \\ B &\to (V_1,\ V_2). \end{aligned}$$

The last missing piece of a grammar that could generate our example is to define notes to be played in mentioned melodic sections. Sonata rules for the first two measures would look like

$$(T_1, T_1') \to (d_{[e,2]}h_{[e,1]}a_{[e,2]}c_{[e,2]},\ d_{[e,2]}h_{[e,1]}a_{[e,1]}c_{[e,2]}),$$

$$(T_1, T_1') \to (h_{[e,1]}a_{[e,-1]}p_{[e,-1]}c_{[e,1]},\ h_{[e,1]}a_{[e,-2]}p_{[e,-1]}c_{[e,2]}).$$

On the right-hand side of the grammar rules, tone names are indexed using brackets, where the first symbol (e) indicates note duration (length—in this case, an eighth note), and the second number specifies the pitch interval or position within the current musical context.

For simplicity this model, is not meant to analyze the musical structure beyond the level of a single measure. This approach helps to ensure rhythmic consistency in the generated music and provides a clearer, more polished grammatical representation. Additionally, it eliminates the need to calculate the exact number of beats per measure or manage the filling of any remaining rhythmic gaps. The presented approach could be applied to any musical piece. We define our form, and after that, from form, we can generate various numbers of melodic and harmonic passages. Formally, this can be represented by grammar rules of the following general structure:

$$(A, A') \to (T_1H_1T_2H_2,\ T_1'H_1'T_2'H_2'),$$
$$B \to (V_1H_1,\ V_2H_2).$$

Here, we have a characterization of a musical piece that features a switch between tonic and harmonic sections. Followed by different variations that could be picked up from classical composers like Bach, Beethoven and others. This is a creative process, and it is up to the creator of the grammar to determine how their music is perceived.

Encoding Multi-Instrumental Compositions into Grammar Rules

We have covered how to create a musical piece when there is only one instrument and needs only one staff. For example, a piano has two staffs. Of course, a staff can still be interpreted by an instrument, but it would lack melody or harmony. From Figure 2, we can see how important it is to have a model that is able to synchronize the generation of music between treble and bass clefs for piano. The bass clef mirrors the melody created by treble clef. For this reason, we use a rule-synchronized model that ensures these properties are preserved. A similar approach can be applied to music for multiple instruments, where instruments often copy the melody, create contrast, create tension, or use other musical expressions to make music interesting.

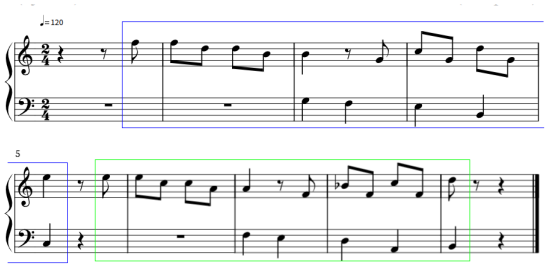

Figure 2: A small example of dependencies between music staffs.

Figure 2 comes from the development of [23]. The first rectangle (green) is a variation of the notes selected in the second rectangle (blue). This can be easily encoded into a 2-component system:

$$G_s = (G_1, G_2, Q),$$

where

- $G_1 = (\{S_1, T, T_\downarrow\}, \{r_{[-,q,-]}, r_{[-,e,-]}, f_{[-,e,2]}, d_{[-,e,2]}, h_{[-,e,1]}, f_{[\downarrow,e,2]}, d_{[\downarrow,e,2]}, h_{[\downarrow,e,1]}, h_{[-,q,1]}, g_{[-,e,1]}, c_{[-,e,2]}, c_{[\downarrow,e,2]}, g_{[\downarrow,e,1]}, h_{[\downarrow,q,1]}, e_{[-,q,2]}, e_{[-,e,2]}, r_{[-,q,-]}\},$
  $\{1\colon S_1 \to (r_{[-,q,-]} r_{[-,e,-]} f_{[-,e,-]} T, T_\downarrow),$
  $2\colon (T, T_\downarrow) \to (f_{[-,e,2]} d_{[-,e,2]} d_{[-,e,2]} h_{[-,e,1]} T, f_{[\downarrow,e,2]} d_{[\downarrow,e,2]} d_{[\downarrow,e,2]} h_{[\downarrow,e,1]} T_\downarrow),$
  $3\colon (T, T_\downarrow) \to (h_{[-,q,1]} r_{[-,e,-]} g_{[-,e,1]} T, h_{[\downarrow,q,1]} r_{[-,e,-]} g_{[\downarrow,e,1]} T_\downarrow),$
  $4\colon (T, T_\downarrow) \to (c_{[-,e,2]} g_{[-,e,1]} d_{[-,e,2]} g_{[-,e,1]} T, c_{[\downarrow,e,2]} g_{[\downarrow,e,1]} d_{[\downarrow,e,2]} g_{[\downarrow,e,1]} T_\downarrow),$
  $5\colon (T, T_\downarrow) \to (e_{[-,q,2]} r_{[-,e,-]} e_{[-,e,2]}, e_{[-,e,2]} r_{[-,e,-]} r_{[-,q,-]})\}, S_1)$
- $G_2 = (\{S_2, B, B_\downarrow\}, \{r_{[-,h,-]}, g_{[-,q,-]}, f_{[-,q,-]}, g_{[\downarrow,q,-]}, f_{[\downarrow,q,-]}, e_{[-,q,-]}, h_{[-,q,-]}, e_{[\downarrow,q,-]}, h_{[\downarrow,q,-]}, a_{[-,q,-]}, r_{[-,q,-]}, a_{[\downarrow,q,-]}, r_{[-,q,-]}\},$
  $\{1\colon S_2 \to (r_{[-,h,-]} B, B_\downarrow),$
  $2\colon (B, B_\downarrow) \to (r_{[-,h,-]} B, r_{[-,h,-]} B_\downarrow)\}$
  $3\colon (B, B_\downarrow) \to (g_{[-,q,-]} f_{[-,q,-]} B, g_{[\downarrow,q,-]} f_{[\downarrow,q,-]} B_\downarrow)$
  $4\colon (B, B_\downarrow) \to (e_{[-,q,-]} h_{[-,q,-]} B, e_{[\downarrow,q,-]} h_{[\downarrow,q,-]} B_\downarrow),$
  $5\colon (B, B_\downarrow) \to (a_{[-,q,-]} r_{[-,q,-]}, a_{[\downarrow,q,-]} r_{[-,q,-]})\}, S_2)$
- $Q = \{(1,1), (2,2), (3,3), (4,4), (5,5)\}$.

This shows how easy it is to encode one of the most popular classical songs into the grammar. Grammar G_1 has rules that can be applied to generate the treble clef for piano and G_2 produces the bass clef. Each measure for both treble and bass clefs is synchronized in the set Q.

Derivation Process in Multi-Generative Grammar

With the intention to create a music piece, rules have to be applied in a certain order. First, we rewrite starting symbol with nonterminals to define structure of the composition. With that, we can start to rewrite structure symbols so that final melodies and harmonies can take the form.

To illustrate this, let us begin with an example that generates jazz music for piano using both the treble and bass clef:

$$G_s = (G_1, G_2, Q),$$

in which

- $G_1 = (\{S_1, A, B\}, \{c_y, a_x, g_x, e_y, f_x, \alpha_z, \beta_z, \gamma_z, \delta_z, \varepsilon_w, \zeta_w\},$
  $\{1\colon S_1 \to (AABAS_1), 2\colon S_1 \to (AABA), 3\colon (A,A,A) \to (MH, MH, MH),$
  $4\colon (M,M,M) \to (c_y c_y a_x g_x, c_y c_y a_x g_x, c_y c_y a_x g_x),$
  $5\colon (H,H,H) \to (\alpha_z \beta_z \gamma_z \delta_z, \alpha_z \beta_z \gamma_z \delta_z, \alpha_z \beta_z \gamma_z \delta_z),$
  $6\colon B \to (M_1 H_1), 7\colon (M_1, H_1) \to (\varepsilon_w \zeta_w, e_y a_x f_x a_x)\}, S_1),$
- $G_2 = (\{S_2, A, B, P, L\}, \{c_v, g_v, r_v, a_u, e_v, r_t\},$
  $\{1\colon S_2 \to (AABAS_2), 2\colon S_2 \to (AABA), 3\colon (A,A,A) \to (PL, PL, PL),$
  $4\colon (P,P,P) \to (c_v g_v c_v g_v r_v g_v c_v g_v, c_v g_v c_v g_v r_v g_v c_v g_v, c_v g_v c_v g_v r_v g_v c_v g_v),$
  $5\colon (L,L,L) \to (c_v a_u e_s r_v r_t e_s a_u, c_v a_u e_s r_v r_t e_s a_u, c_v a_u e_s r_v r_t e_s a_u),$
  $6\colon B \to (PL), 7\colon (P,L) \to (c_v g_v c_v g_v r_v g_v c_v g_v, c_v a_u e_s r_v r_t e_s a_u)\}, S_2),$

- $Q = \{(1,1),(2,2),(3,3),(4,4),(5,5),(6,6),(7,7)\}$.

Grammars in system G_s use substitution of token symbols for better readability in defined grammar and in following derivations. Explanation of the tokens in G_1 is in the tables Tab. 1 and Tab. 2.

These tables explain the symbols used in G_s in this section. The first position, for example in $c_{-,q,2}$ is used for a variation technique that moves the tone. The symbol q represents a quarter note, e an eighth note, and h a half note. The last element specifies the octave in which the note is placed.

Symbol	Note or Chord
c_y	$c_{[-,q,2]}$
a_x	$a_{[-,q,1]}$
g_x	$g_{[-,q,1]}$
e_y	$e_{[-,q,2]}$
f_x	$f_{[-,q,1]}$
α_z	$Chord(C,C_2,E)_{[-,q,1]}$
β_z	$Chord(D,F,A)_{[-,q,1]}$
γ_z	$Chord(A,C,F)_{[-,q,1]}$
δ_z	$Chord(A,C,E)_{[-,q,1]}$
ε_w	$Chord(A,C,F)_{[-,h,1]}$
ζ_w	$Chord(F,A,C)_{[-,h,1]}$

Table 1: Mapping of terminal symbols to musical feature vectors of G_1.

Symbol	Note
c_v	$c_{[-,e,1]}$
g_v	$g_{[-,e,1]}$
r_t	$r_{[-,e,1]}$
a_u	$a_{[\flat,e,1]}$
e_v	$e_{[\flat,e,1]}$
r_t	$r_{[-,e,1]}$

Table 2: Mapping of terminal symbols to musical feature vectors of G_2.

For this G_s, we can create the following derivation steps:

- $(S_1,S_2) \Rightarrow^1 (AABA,AABA) \Rightarrow^2 (MHMHBMH,PLPLBPL)$
  $\Rightarrow^3 (MHMHM_1H_1MH,PLPLPLPL)$
  $\Rightarrow^4 (c_yc_ya_xg_xHc_yc_ya_xg_xHM_1H_1c_yc_ya_xg_xH,$
  $c_vg_vc_vg_vr_vg_vc_vg_vLc_vg_vc_vg_vr_vg_vc_vg_vLPLc_vg_vc_vg_vr_vg_vc_vg_vL)$
  $\Rightarrow \ldots$

Instead of writing out terminal symbols, it is much more interesting to demonstrate terminal symbols already in the music staff. Nonterminal symbols are blank bars that represent the structure. Fig. 3 describes the correspondence between the fifth and sixth derivation steps in G_s and their musical interpretation. More specifically, during $\Rightarrow^5$, G_s rewrites nonterminals H and L from G_1 and G_2, respectively; as a result, all A parts are completed. During $\Rightarrow^6$, G_s completes the generation of the sentence and, therefore, its corresponding musical piece by filling in the missing part of the generated score.

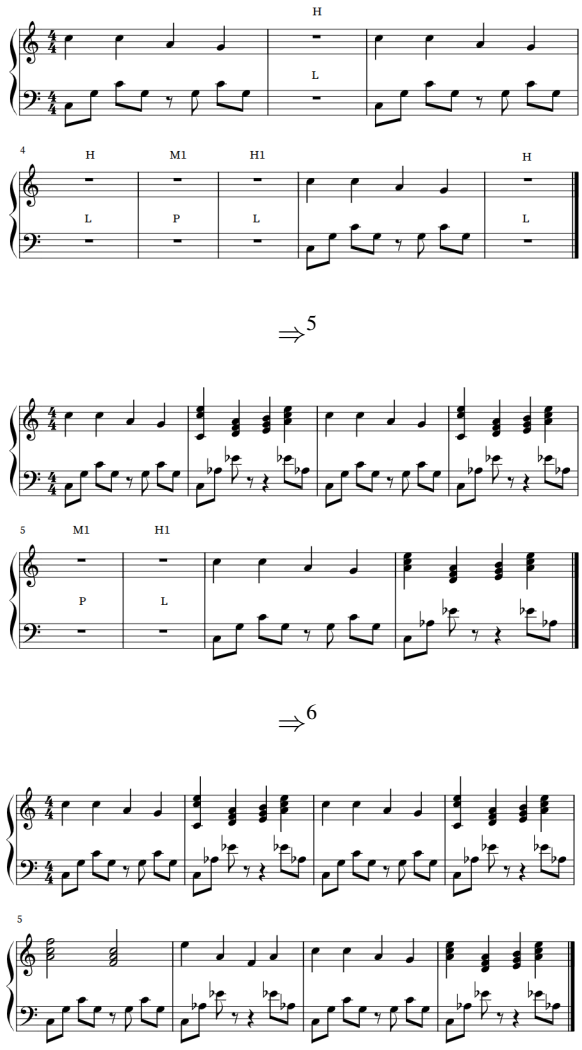

Figure 3: The fifth and sixth derivation step shown in music staff that corresponds to $(5,5) \in Q$, and $(7,7) \in Q$.

5 Example

Until now, we have been generating music for only one instrument. Finally, we will show how our model could generate jazz music. This music is going to be interpreted by a piano and saxophone. The music will take jazz from AABA and will be generated in three strings, two for piano and one for saxophone. So far, we have used variation, tone duration, and tone octave for our generated tokens. Now, we will also incorporate dynamics. An example of a grammar system generating such computation follows:

$$G_s = (G_1, G_2, G_3, Q),$$

in which

- $G_1 = (\{S_1, M_1, M_2, A, B, N\}, \{f_{[-,q,1,-]}, c_{[-,q,1,-]}, f_{[\downarrow,q,1,p]}, c_{[\downarrow,q,1,p]}, g_{[-,q,2,-]},$

$d_{[-,q,2,-]}, g_{[\downarrow,q,2,p]}, d_{[\downarrow,q,2,p]}, e_{[-,h,2,-]}, g_{[-,h,2,-]}, e_{[\downarrow,h,2,p]}, g_{[\downarrow,h,2,p]}, f_{[-,h,2,-]}$,
$a_{[-,h,2,-]}, f_{[\downarrow,h,2,p]}, a_{[\downarrow,h,2,p]}\}$,
$\{1\colon S \to (AABA)$,
$2\colon (A,A,A) \to (M_1M_2M_2M_1, M_1M_2M_2M_1, M_1M_2M_2M_1)$,
$3\colon (M_1,M_1,M_1) \to (f_{[-,q,1,-]}c_{[-,q,1,-]}c_{[-,h,1,-]}$,
$f_{[\downarrow,q,1,p]}c_{[\downarrow,q,1,p]}c_{[\downarrow,h,1,p]}, f_{[-,q,1,-]}c_{[-,q,1,-]}c_{[-,h,2,-]})$,
$4\colon (M_1,M_1,M_1) \to (g_{[-,q,2,-]}d_{[-,q,2,-]}d_{[-,h,2,-]}$,
$g_{[\downarrow,q,2,p]}d_{[\downarrow,q,2,p]}d_{[\downarrow,h,2,p]}, g_{[-,q,2,-]}d_{[-,q,2,-]}d_{[-,h,2,-]})$,
$5\colon (M_2,M_2,M_2) \to (e_{[-,h,2,-]}g_{[-,h,2,-]}, e_{[\downarrow,h,2,p]}g_{[\downarrow,h,2,p]}, e_{[-,h,2,-]}g_{[-,h,2,-]})$,
$6\colon (M_2,M_2,M_2) \to (f_{[-,h,2,-]}a_{[-,h,2,-]}, f_{[\downarrow,h,2,p]}a_{[\downarrow,h,2,p]}, f_{[-,h,2,-]}a_{[-,h,2,-]})$,
$7\colon B \to (NNNN)$
$8\colon N \to (r_{[-,f,-,-]})\})$,

- $G_2 = (\{S_2,A,B,P,R,N\}, \{\gamma_{[-,h,1,-]}, \gamma_{[P,h,1,-]}, \gamma_{[R,h,1,p]}, r_{[-,f,-,-]}\}$,
$\{1\colon S \to (AABA)$,
$2\colon (A,A,A) \to (PRPR, PRPR, PRPR)$,
$3\colon (P,P,P) \to (\gamma_{[-,h,1,-]}\gamma_{[P,h,1,-]}R, \gamma_{[-,h,1,p]}\gamma_{[P,h,1,p]}R, \gamma_{[-,h,1,-]}\gamma_{[P,h,1,-]}R)$,
$4\colon (R,R,R) \to (\gamma_{[-,h,1,-]}\gamma_{[R,h,1,-]}P, \gamma_{[-,h,1,p]}\gamma_{[R,h,1,p]}P, \gamma_{[-,h,1,-]}\gamma_{[R,h,1,-]}P)$,
$5\colon (P,P,P) \to (\gamma_{[-,h,1,-]}\gamma_{[P,h,1,-]}, \gamma_{[-,h,1,p]}\gamma_{[P,h,1,p]}, \gamma_{[-,h,1,-]}\gamma_{[P,h,1,-]})$,
$6\colon (R,R,R) \to (\gamma_{[-,h,1,-]}\gamma_{[R,h,1,-]}, \gamma_{[-,h,1,p]}\gamma_{[R,h,1,p]}, \gamma_{[-,h,1,-]}\gamma_{[R,h,1,-]})$,
$7\colon B \to (NNNN)$
$8\colon N \to (r_{[-,f,-,-]})\})$,
- $G_3 = (\{S_3,A,B,H,M_{31},M_{32}\}, \{e_{[-,h,1,-]}, g_{[-,h,1,-]}, a_{[-,h,1,-]}, f_{[-,h,1,-]}$,
$h_{[-,h,1,-]}, c_{[-,q,1,-]}\}$,
$\{1\colon S \to (AABA)$,
$2\colon (A,A,A) \to (MMMM, MMMM, MMMM)$,
$3\colon (M,M,M) \to (e_{[-,h,1,-]}g_{[-,h,1,-]}, e_{[\uparrow,h,1,-]}g_{[\uparrow,h,1,-]}, e_{[-,h,1,-]}g_{[-,h,1,-]})$,
$4\colon (M,M,M) \to (a_{[-,h,1,-]}f_{[-,h,1,-]}, a_{[\uparrow,h,1,-]}f_{[\uparrow,h,1,-]}, a_{[-,h,1,-]}f_{[-,h,1,-]})$,
$5\colon (M,M,M) \to (g_{[-,h,1,-]}e_{[-,h,1,-]}, g_{[\uparrow,h,1,-]}e_{[\uparrow,h,1,-]}, g_{[-,h,1,-]}e_{[-,h,1,-]})$,
$6\colon (M,M,M) \to (f_{[-,h,1,-]}f_{[-,h,1,-]}, f_{[\uparrow,h,1,-]}f_{[\uparrow,h,1,-]}, f_{[-,h,1,-]}f_{[-,h,1,-]})$,
$7\colon B \to (HM_{31}M_{32}H)$
$8\colon (H,H) \to (\alpha_{[-,h,1,-]}\beta_{[-,h,1,-]}, \alpha_{[r,h,1,-]}\beta_{[r,h,1,-]})$
$9\colon M_{31} \to (e_{[-,h,1,-]}g_{[-,h,1,-]}a_{[-,h,1,-]}h_{[-,h,1,-]})$
$10\colon M_{32} \to (h_{[-,q,1,-]}c_{[-,q,1,-]}a_{[-,q,1,-]}f_{[-,q,1,-]})\})$,
- $Q = \{(1,1,1),(2,2,2),(3,3,3),(3,5,3),(4,4,4),(4,6,4),(5,4,5),(5,6,5),$
$(6,3,6),(6,5,6),(7,7,7),(8,8,8),(8,8,9),(8,8,10)\}$.

A composition that could be generated by the presented grammar system is shown in Fig. 4. It shows that grammar can generate meaningful music with various music techniques. To describe what is in the figure, we would start with the piano part. In the piano part, the A section of the composition presents the main theme and completes the harmony in the treble clef, while additional harmonic support is found in the bass clef. Alongside the piano, the saxophone is there to provide a second harmonic party to enrich the melody. The role of the Sax is to create an interesting contrast to the main melody. While the primary theme ascends, the Sax line moves downwards, which creates a playful tension and enriches the overall texture. A bridge is created by Sax solo, which is an alternation between harmonic and melodic material to create contrast with the A sections and a bridge between the piano part of the main theme and the last repetition of the main theme that ends the composition.

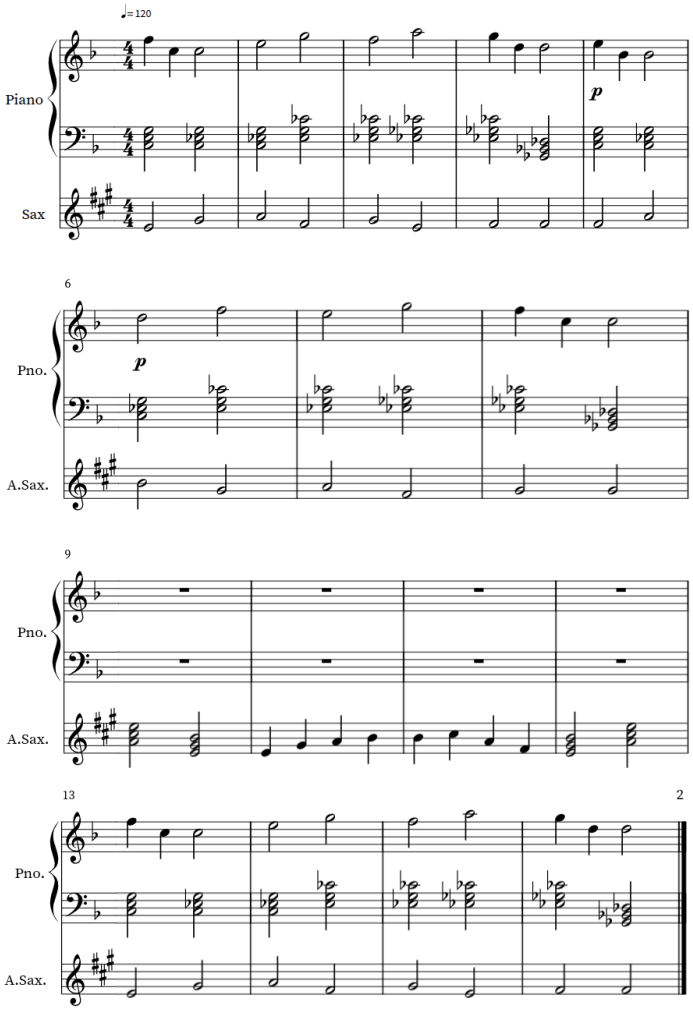

Figure 4: Illustrative example of multi-instrument jazz composition.

Tables 3 and 4 show interpretation of symbols from G_s in this section.

Symbol	Note
γ	$Chord(C,E,G)$
γ	$Chord(C,Es,G)$
γ	$Chord(Es,G,Ces)$
γ	$Chord(Es,Ges,Ces)$
γ	$Chord(Ges,Hes,Des)$

Table 3: Mapping of terminal symbols to chords from Tonnetz [8] walk using PR transformations of G_2.

Symbol	Chord
α	$Chord(A,C,E)$
β	$Chord(E,G,H)$

Table 4: Mapping of terminal symbols to musical feature vectors of G_3.

To see more song examples and implementation details visit our GitHub repository.[1]

[1] Implementation details at https://github.com/NaKamize/music-grammar-system

6 Evaluation

We mentioned that music is a creative process, and because of that, it is difficult to find a mathematical formula that provides a number or graph to help compare our method to existing algorithms for music generation. And we don't need that. The biggest advantage is the enforcement of the rules and their synchronization, which allows the music structure to fit its nature perfectly. We showed this through the provided examples. Generated examples keep the musical structure as it was intended and follow the rules of music theory. This is due to the correctly selected rules. The playable sound examples are stored in GitHub[1] with the implementation and implementation details.

To compare our method to L-systems, we are able to generate not just the fractal music but any music that has structure. We don't require postprocessing of the generated string; it can be interpreted instantly. Probabilistic formal models have the advantage that they can learn to imitate any style and generate that style of music. In comparison, our method is as good as the person who is creating the rules. The tone rules have to fit a specific style or melody.

This method is great at creating synchronized multi-instrument pieces, and its use could be in the procedural generation of music for computer games, as [6]. There have been several attempts to enhance music generation using neural networks. However, they often struggle to capture long-term dependencies or musical structure. A hybrid approach that combines them with our model could be advantageous. Those approaches keep the rich and expressive sound of neural networks and combine it with the needed structure and dependencies.

7 Conclusion

To summarize the present application-oriented paper as simply as possible, we have demonstrated how to orchestrate music by using grammar systems (see Section 3 and 4). In addition, we have illustrated an orchestration of this kind by an example (see Section 5).

Although we have described this kind of orchestration in a rather great detail, there still remain many open problem areas related to the subject of this paper. Next, we suggest five of them.

(1) Investigate classical topics of formal language theory, such as decidable problems or closure properties, in terms of the systems from Section 3.

(2) Conceptualize, re-formulate and investigate the subject of this paper in terms of other language models, such as jumping or regulated grammars and automata (see [20, 21]).

(3) Restrict the systems from Section 3 so they can use only context-free or even linear rules. What kind of music can be orchestrated by systems restricted in this way?

(4) Many compositions for orchestras frequently contain long musical passages during which several instruments simultaneously play the same music. Can the grammar systems considered in Section 3 be modified so that a single component produce a score for all these instruments, which play the same music? Even more generally, can these systems be modified so that a single component produces scores for several instruments, possibly playing different music?

(5) Consider only smaller-sized orchestras, such as chamber orchestras. What are the simplest possible versions of the grammar systems that can orchestrate them?

Acknowledgments

This work was supported by Brno University of Technology grant FIT-S-23-8209.

References

[1] Andrew Adamatzky & Genaro J. Martínez, editors (2016): *Designing Beauty: The Art of Cellular Automata*, 1st edition. *Emergence, Complexity and Computation* 20, Springer, doi:10.1007/978-3-319-32922-7. Kindle Edition.

[2] A.V. Aho & J.D. Ullman (1972): *The Theory of Parsing, Translation, and Compiling*. Prentice-Hall, Series in Automatic Computation.

[3] David D. Albarracín-Molina, Alfredo Raglio, Francisco Rivas-Ruiz & Francisco J. Vico (2021): *Using Formal Grammars as Musical Genome*. *Applied Sciences* 11(9), p. 4151, doi:10.3390/app11094151.

[4] Bernard Bel & Jim Kippen (1992): *Modelling music with grammars: formal language representation in the Bol Processor*. In: *Computer Representations and Models in Music*, Academic Press, pp. 207–238. Available at `https://shs.hal.science/halshs-00004506`.

[5] Michael Edwards (2011): *Algorithmic Composition: Computational Thinking in Music*. *Communications of the ACM* 54(7), pp. 58–67, doi:10.1145/1965724.1965742.

[6] Lukas Eibensteiner (2018): *Procedural Music Generation with Grammars*. In: *Proceedings of the 22nd Central European Seminar on Computer Graphics (CESCG)*.

[7] Édouard Gilbert & Darrell Conklin (2007): *A Probabilistic Context-Free Grammar for Melodic Reduction*. In: *Proceedings of the International Workshop on Artificial Intelligence and Music*, Hyderabad, India, pp. 83–94.

[8] Michael Gogins (2006): *Score Generation in Voice-Leading and Chord Spaces*. In Georg Essl & Ichiro Fujinaga, editors: *Proceedings of the 2006 International Computer Music Conference (ICMC)*, International Computer Music Association, pp. 455–457.

[9] M. A. Harrison (1978): *Introduction to Formal Language Theory*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[10] David Humphreys, Kirill Sidorov, Andrew Jones & David Marshall (2021): *An Investigation of Music Analysis by the Application of Grammar-Based Compressors*. *Journal of New Music Research* 50(4), pp. 312–341, doi:10.1080/09298215.2021.1978505.

[11] Zeyu Jin & Roger B. Dannenberg (2013): *Formal Semantics for Music Notation Control Flow*. In: *Proceedings of the International Computer Music Conference (ICMC)*. Available at `http://hdl.handle.net/2027/spo.bbp2372.2013.010`.

[12] Bryan Jurish (2004): *Music as a Formal Language*. In Fränk Zimmer, editor: *bang | pure data*, Wolke Verlag, Hofheim, pp. 45–58.

[13] Tim Kadlec, Ivica Gabrišová, Janka Jámborová, Michal Vojáček, Emily Beynon, Robert Heger & Halka Klánská (2022): *Methodology: Increasing the Efficiency and Quality of Instrumentalists' Preparation for Orchestral Auditions*. Accessed: 2025-03-14.

[14] Robert M. Keller & David R. Morrison (2007): *A Grammatical Approach to Automatic Improvisation*. In: *Proceedings of the 4th Sound and Music Computing Conference (SMC)*, Lefkada, Greece, pp. 330–337.

[15] Roman Lukáš (2006): *Multigenerative Grammar Systems*. Ph.d. dissertation, Brno University of Technology, Brno, Czech Republic. Supervisor: Prof. RNDr. Alexander Meduna, CSc.

[16] Stelios Manousakis (2006): *Musical L-Systems*. Master's thesis, Royal Conservatory, The Hague.

[17] Stelios Manousakis (2009): *Non-Standard Sound Synthesis with L-Systems*. *Leonardo Music Journal* 19, pp. 85–94, doi:10.1162/lmj.2009.19.85.

[18] Jon McCormack (1996): *Grammar-Based Music Composition*. In A. Stocker, M. Schenker & L. M. Browne, editors: *Complex Systems: From Local Interactions to Global Phenomena*, 96, IOS Press, pp. 321–336.

[19] Alexander Meduna, Petr Horáček & Martin Tomko (2020): *Handbook of Mathematical Models for Languages and Computation*. Computing and Networks, The Institution of Engineering and Technology, London, UK. Kindle Edition.

[20] Alexander Meduna & Zbyněk Křivka (2024): *Jumping Computation: Updating Automata and Grammars for Discontinuous Information Processing*. CRC Press, Boca Raton.

[21] Alexander Meduna & Petr Zemek (2014): *Regulated Grammars and Automata*. Springer, New York, doi:10.1007/978-1-4939-0369-6.

[22] Orestis Melkonian (2019): *Music as Language: Putting Probabilistic Temporal Graph Grammars to Good Use*. In: *Proceedings of the 7th ACM SIGPLAN International Workshop on Functional Art, Music, Modeling, and Design (FARM 2019)*, Association for Computing Machinery, pp. 1–10, doi:10.1145/3331543.3342576.

[23] Tom Pankhurst: *Sonata Form*. `https://alevelmusic.com/alevelcompositionhelp/composing-help/sonata-form-2/sonata-form/`. Accessed: 2025-03-15.

[24] Przemyslaw Prusinkiewicz (1986): *Score generation with L-systems*. In: *Proceedings of the International Computer Music Conference (ICMC)*, pp. 455–457.

[25] Leonardo Ravelli (2025): *Understanding music to improvise better: Form in jazz standards*. Accessed: 2025-03-14.

[26] Sergio Krakowski Costa Rego (2009): *Rhythmically-Controlled Automata Applied to Musical Improvisation*. Ph.d. dissertation, Instituto Nacional de Matemática Pura e Aplicada (IMPA), Rio de Janeiro, Brazil.

[27] Ana Rodrigues, Ernesto Costa, Amílcar Cardoso, Penousal Machado & Tiago Cruz (2016): *Evolving L-Systems with Musical Notes*. In Colin Johnson, Alvaro Carballal & João Correia, editors: *Evolutionary and Biologically Inspired Music, Sound, Art and Design*, *Lecture Notes in Computer Science* 9596, Springer International Publishing, pp. 186–201, doi:10.1007/978-3-319-31008-4_-13.

[28] G. Rozenberg & A. Salomaa (1997): *Handbook of Formal Languages, Vol. 1: Word, Language, Grammar*. Springer-Verlag, New York, doi:10.1007/978-3-642-59126-6.

[29] A. Salomaa (1973): *Formal Languages*. Academic Press, London.

[30] Peter Worth & Susan Stepney (2005): *Growing Music: Musical Interpretations of L-Systems*. In Franz Rothlauf et al., editors: *Applications of Evolutionary Computing*, *Lecture Notes in Computer Science* 3449, Springer, pp. 545–550, doi:10.1007/978-3-540-32003-6_56.

[31] Willem Zuidema, Dieuwke Hupkes, Geraint A. Wiggins, Constance Scharff & Martin Rohrmeier (2018): *Formal Models of Structure Building in Music, Language, and Animal Song*. In Henkjan Honing, editor: *The Origins of Musicality*, MIT Press, pp. 253–286, doi:10.48550/arXiv.1901.05180.

On Repetitive Finite Automata with Translucent Words

František Mráz
Faculty of Mathematics and Physics
Charles University, Malostranské nám. 25
118 00 Praha 1, Czech Republic
frantisek.mraz@mff.cuni.cz

Friedrich Otto
Fachbereich Elektrotechnik/Informatik
Universität Kassel
34109 Kassel, Germany
f.otto@uni-kassel.de

We introduce and study the repetitive variants of the deterministic and the nondeterministic finite automaton with translucent words (DFAwtw and NFAwtw). On seeing the right sentinel, a repetitive NFAwtw need not halt immediately, accepting or rejecting, but it may change into another state and continue with its computation. We establish that a repetitive DFAwtw already accepts a language that is not even semi-linear, which shows that the property of being repetitive increases the expressive capacity of the DFAwtw and the NFAwtw considerably.

Keywords: Finite automaton – translucent word – language class – hierarchy – closure property – emptiness problem

1 Introduction

The deterministic and the nondeterministic finite automaton *with translucent letters* (or DFAwtl and NFAwtl) was introduced by Nagy and Otto in [12] (see also [19]) as a reinterpretation of certain cooperating distributed systems of a very restricted type of deterministic restarting automata. For each state q of an NFAwtl, there is a set $\tau(q)$ of *translucent letters*, which is a subset of the input alphabet that contains those letters that the automaton cannot see when it is in state q. Accordingly, in each step, the NFAwtl just reads (and deletes) the first letter from the left that it can see, that is, which is not translucent for the current state. It has been shown that the NFAwtls accept a class of semi-linear languages that properly contains all rational trace languages, whereas its deterministic variant, the DFAwtl, is properly less expressive. In fact, the DFAwtl accepts a class of languages that is incomparable to the rational trace languages with respect to inclusion [11, 13, 14, 15]. In addition, while the obvious upper bound for the time complexity of the membership problem for a DFAwtl is DTIME(n^2), an improved upper bound of DTIME($n \cdot \log n$) is derived in [10].

In [6], the authors present a variant of the finite automaton with translucent letters which, after reading and deleting a letter, does not return its head to the left end of its tape, but that rather continues from the position of the letter just deleted. When the end-of-tape marker is reached, this automaton can decide whether to accept, reject, or continue with its computation, which means that it changes its state and again reads the remaining tape contents from the beginning. The latter property of the automaton is called 'repetitiveness'. This type of automaton, called a *non-returning finite automaton with translucent letters* or an *nrNFAwtl*, is strictly more expressive than the NFAwtl. This result also holds for the deterministic case, although the deterministic variant, the *nrDFAwtl*, is still not sufficiently expressive to accept all rational trace languages.

In [7], the nrDFAwtl and the nrNFAwtl are compared to the jumping finite automaton, the right one-way jumping finite automaton of [1, 3], and the right-revolving finite automaton of [2], deriving the complete taxonomy of the resulting classes of languages.

N. Moreira and L. Prigioniero (Eds.): 15th International Workshop on Non-Classical Models of Automata and Applications (NCMA 2025).
EPTCS 422, 2025, pp. 59–72, doi:10.4204/EPTCS.422.5

While an NFAwtl halts immediately when it sees its end-of-tape marker, either accepting or rejecting, a non-returning NFAwtl as described above is repetitive, that is, it may continue its computation in the corresponding situation. In [8], the authors study the influence that this property has on automata with translucent letters. As it turns out, NFAwtls that are repetitive are equivalent to NFAwtls that are non-repetitive, while the repetitive DFAwtls are strictly more expressive than the DFAwtls that are not repetitive. On the other hand, nondeterministic and deterministic finite automata with translucent letters that are non-returning and non-repetitive accept just the regular languages. That is, they are equivalent to finite automata without translucent letters. A recent survey on the various types of automata with translucent letters can be found in [18].

Finally, in [16], the finite automaton with translucent letters is generalized by extending the sets of translucent letters to sets of translucent words, which yields the *finite automaton with translucent words* or *NFAwtw*. An NFAwtw reads (and deletes) the first letter from the left that is only preceded by a prefix that is a product of words that are translucent for the current state. This gives the automaton more control over the structure of the prefix ignored in a transition than for an NFAwtl. In order to guarantee that the resulting computation relation of an NFAwtw can be computed efficiently, the following two technical restrictions have been placed on the set $\tau(q)$ of translucent words associated with a state q of an NFAwtw A:

- the set of translucent words $\tau(q)$ is a finite prefix code, and
- no word in the set $\tau(q)$ may begin with a letter a that the NFAwtw A can read in state q, that is, for which A has a possible transition of the form $q' \in \delta(q,a)$.

Together these restrictions imply that the first letter from the left that an NFAwtw can read in a state q can be determined by simply scanning the current tape contents letter by letter from left to right. It turned out that there are languages that are accepted by deterministic finite automata with translucent words (that is, by *DFAwtws*), but that are not even accepted by any nondeterministic finite automata with translucent letters.

The finite automaton with translucent words can be parameterized by placing two restrictions on the size of the sets of translucent words admitted:

1. An NFAwtw A is k-cardinality-restricted for some integer $k \geq 1$, if each set of translucent words of A contains at most k elements.
2. An NFAwtw A is ℓ-length-restricted for some integer $\ell \geq 1$, if no set of translucent words of A contains a word of length larger than ℓ.

Obviously, the 1-length-restricted NFAwtw is just the NFAwtl, and moreover, the notion of cardinality-restriction carries over to the NFAwtl. These two parameters induce infinite strictly ascending two-dimensional hierarchies of language classes for the NFAwtw and as well as for the DFAwtw [17]. In fact, the hierarchy based on cardinality-restriction alone and the hierarchy based on length-restriction alone both carry over to the case of binary alphabets [9].

Here, we define and study the repetitive variants of the NFAwtw and its deterministic variant, the DFAwtw. On seeing the end-of-tape marker, such an automaton may either halt, accepting or rejecting, or it may change its state and reposition its head on the first letter of the current tape contents, continuing with its computation.

The following important results are derived:

- There exists a repetitive DFAwtw that accepts a language which is not semi-linear (Theorem 15).
- There exists a language that is accepted by a repetitive NFAwtw, but not by any repetitive DFAwtw (Theorem 18).

- For repetitive DFAwtws, emptiness is undecidable (Theorem 20). Moreover, finiteness, regularity, inclusion, equivalence, and boundedness are undecidable for this type of automaton, too.

However, closure and non-closure properties for the various classes of repetitive NFAwtws and DFAwtws have not yet been determined.

2 Definitions and Known Results on Finite Automata with Translucent Words

First we restate the definition of the finite automaton with translucent words as defined in [16]. However, we slightly change the definition by removing the final states and by adjusting the definition of the transition function accordingly. Here we use $\mathscr{P}(S)$ to denote the powerset of a set S and $\mathscr{P}_{\text{fin}}(S)$ to denote the set of all finite subsets of S.

Definition 1 *A* finite automaton with translucent words*, or an* NFAwtw*, is defined by a 6-tuple*

$$A = (Q, \Sigma, \lhd, \tau, I, \delta),$$

where Q is a finite set of states, Σ is a finite input alphabet, $\lhd \notin \Sigma$ is a special letter that serves as an end-of-tape marker, $I \subseteq Q$ is a set of initial states, $\tau : Q \to \mathscr{P}_{\text{fin}}(\Sigma^*)$ is a translucency mapping*, and*

$$\delta : Q \times (\Sigma \cup \{\lhd\}) \to \mathscr{P}(Q) \cup \{\mathsf{Accept}, \mathsf{Reject}\}$$

is a transition function. Here we require that, for each state $q \in Q$ and each letter $a \in \Sigma$, $\delta(q,a) \subseteq Q$ and $\delta(q,\lhd) \in \{\mathsf{Accept}, \mathsf{Reject}\}$. The latter means that, on seeing the sentinel $\lhd$, the NFAwtw A halts immediately, either accepting or rejecting.

For each state $q \in Q$, let $\Sigma_q^{(A)} = \{\, a \in \Sigma \mid \delta(q,a) \neq \emptyset \,\}$, that is, $\Sigma_q^{(A)}$ contains those letters that A can read in state q. It is required that the set of translucent words $\tau(q)$ satisfies the following two restrictions:

- *If $\tau(q) \neq \emptyset$, then $\tau(q)$ is a finite prefix code.*
- *No word in $\tau(q)$ begins with a letter from the set $\Sigma_q^{(A)}$.*

Actually, this means that the set $\tau(q) \cup \Sigma_q^{(A)}$ is a finite prefix code.

The computation relation $\vdash_A^*$ that A induces on its set of configurations $Q \cdot \Sigma^* \cdot \lhd \cup \{\mathsf{Accept}, \mathsf{Reject}\}$ is the reflexive and transitive closure of the following single-step computation relation, where $q \in Q$ and $w \in \Sigma^*$:

$$qw \cdot \lhd \vdash_A \begin{cases} q'uv \cdot \lhd, & \text{if } w = uav,\ u \in (\tau(q))^*,\ a \in \Sigma_q^{(A)},\ v \in \Sigma^*, \text{ and } q' \in \delta(q,a), \\ \mathsf{Reject}, & \text{if } w = uav,\ u \in (\tau(q))^*,\ a \in \Sigma \setminus \Sigma_q^{(A)},\ v \in \Sigma^*, \text{ and } av \notin \tau(q) \cdot \Sigma^*, \\ \mathsf{Accept}, & \text{if } w \in (\tau(q))^* \text{ and } \delta(q,\lhd) = \mathsf{Accept}, \\ \mathsf{Reject}, & \text{if } w \in (\tau(q))^* \text{ and } \delta(q,\lhd) = \mathsf{Reject}. \end{cases}$$

A word $w \in \Sigma^*$ is accepted by *A if there exist an initial state $q_0 \in I$ and a computation $q_0 w \cdot \lhd \vdash_A^* \mathsf{Accept}$. Now $L(A)$ denotes the* language accepted by *A and $\mathscr{L}(\mathsf{NFAwtw})$ denotes the class of all languages that are accepted by NFAwtws.*

An NFAwtw $A = (Q, \Sigma, \lhd, \tau, I, \delta)$ is deterministic *(or a* DFAwtw*) if $|I| = 1$ and $|\delta(q,a)| \leq 1$ for each $q \in Q$ and $a \in \Sigma$. For a DFAwtw, we simply replace the set I by the single initial state and write $\delta(q,a) = q'$ instead of $\delta(q,a) = \{q'\}$. Then $\mathscr{L}(\mathsf{DFAwtw})$ denotes the class of all languages that are accepted by DFAwtws.*

As $\tau(q)$ is a prefix code for each state q, the factorization $w = uav$, where $u \in (\tau(q))^*$, $a \in \Sigma$, $v \in \Sigma^*$, and $av \notin \tau(q) \cdot \Sigma^*$, is uniquely determined. This is not the case without the requirement that $\tau(q)$ is a prefix code (see [16]). From the definition of the single step computation relation, we obtain the following property.

Lemma 2 ([16]) *Let $A = (Q, \Sigma, \lhd, \tau, I, \delta)$ be an NFAwtw and assume that $quav \cdot \lhd \vdash_A puv \cdot \lhd$, where $q, p \in Q$, $u \in (\tau(q))^*$, $a \in \Sigma_q^{(A)}$, and $v \in \Sigma^*$. Then $quavw \cdot \lhd \vdash_A puvw \cdot \lhd$ for each word $w \in \Sigma^*$.*

Let $D_1 \subseteq \{a,b\}^*$ be the semi-Dyck language on $\Sigma = \{a,b\}$, that is, D_1 is the language that is generated by the context-free grammar

$$G = (\{S\}, \Sigma, S, \{S \to \lambda, S \to SS, S \to aSb\}).$$

Furthermore, let $\Gamma = \{a,b,c\}$, $\varphi : \Sigma^* \to \Gamma^*$ be the morphism that is defined through $a \mapsto ab$ and $b \mapsto c$, and $L_1 = \varphi(D_1)$.

Lemma 3 ([16]) *The language L_1 is accepted by a DFAwtw, but not by any NFAwtl.*

Each NFAwtl can be extended to an equivalent NFAwtl that only accepts after having read and deleted its input completely (see, e.g., [14]). For NFAwtws, the corresponding technical result holds as well.

Lemma 4 ([16]) *From a given NFAwtw A, one can construct an NFAwtw B such that $L(B) = L(A)$ and B only accepts once it has read and deleted its input completely.*

The NFAwtw B constructed in the proof of this result is inherently nondeterministic, even if the given NFAwtw A happens to be deterministic. Based on this technical result, the following result has been derived.

Proposition 5 ([16]) *If A is an NFAwtw, then there exists a regular sublanguage R of the language $L(A)$ such that R is letter-equivalent to $L(A)$. In fact, an NFA B for the sublanguage R can effectively be constructed from A.*

Here two languages on the same alphabet are called *letter-equivalent* if they have identical images under the corresponding Parikh mapping (see, e.g. [14]). This result has the following immediate consequence.

Corollary 6 ([16]) *The language accepted by an NFAwtw is semi-linear, that is, its Parikh image is a semi-linear subset of $\mathbb{N}^n$, where n is the cardinality of the underlying alphabet.*

In addition, Proposition 5 implies the following negative result, where L_{lin} denotes the deterministic linear language $L_{\text{lin}} = \{ a^n b^n \mid n \geq 0 \}$.

Proposition 7 ([16]) $L_{\text{lin}} \notin \mathscr{L}(\mathsf{NFAwtw})$.

Observe that $L_{\text{lin}} = L_{\text{eq2}} \cap (a^* \cdot b^*)$, where $L_{\text{eq2}} = \{ w \in \{a,b\}^* \mid |w|_a = |w|_b \} \in \mathscr{L}(\mathsf{DFAwtl})$. Thus, Proposition 7 implies, in particular, that the language classes $\mathscr{L}(\mathsf{DFAwtw})$ and $\mathscr{L}(\mathsf{NFAwtw})$ are not closed under intersection and under intersection with regular languages.

Finally, the DFAwtws have been separated from the NFAwtws. Let

$$L_\vee = \{ w \in \Sigma^* \mid \exists n \geq 0 : |w|_a = n \text{ and } |w|_b \in \{n, 2n\} \},$$

where $\Sigma = \{a,b\}$. The language $L_\vee$ is a rational trace language, and hence, it is accepted by an NFAwtl, but it is not accepted by any DFAwtl [15]. In fact, $L_\vee$ is not even accepted by any DFAwtw, either.

Theorem 8 ([17]) $L_\vee \notin \mathscr{L}(\mathsf{DFAwtw})$.

Hence, we have the following proper inclusion.

Corollary 9 ([16]) $\mathscr{L}(\mathsf{DFAwtw}) \subsetneq \mathscr{L}(\mathsf{NFAwtw})$.

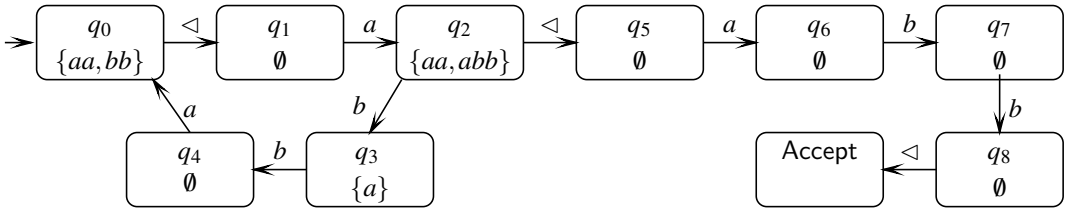

Figure 1: The RDFAwtw $A_{2\text{lin}}$ for the language $L_{2\text{lin}} = \{ a^{2n}b^{2n} \mid n \geq 1 \}$.

3 Repetitive Finite Automata with Translucent Words

Here we present the announced extension of the finite automaton with translucent words.

Definition 10 *Let $A = (Q, \Sigma, \triangleleft, \tau, I, \delta)$ be an NFAwtw.*

(a) *The NFAwtw A is called* repetitive *if, for each state $q \in Q$, $\delta(q, \triangleleft)$ is either a subset of Q or $\delta(q, \triangleleft) \in \{\mathsf{Accept}, \mathsf{Reject}\}$. We use* RNFAwtw (RDFAwtw) *to denote the class of repetitive NFAwtws (DFAwtws). To distinguish the model of Definition 1 from the repetitive NFAwtw, the former is called* non-repetitive.

(b) *The (R)NFAwtw A is k-*cardinality-restricted *(or a k-*r(R)NFAwtw*) for some integer $k \geq 1$, if $|\tau(q)| \leq k$ for each state $q \in Q$. If A is deterministic, then it is called a k-*r(R)DFAwtw.

(c) *The (R)NFAwtw A is ℓ-*length-restricted *(or an ℓ-*lr-(R)NFAwtw*) for some integer $\ell \geq 1$, if $|u| \leq \ell$ for all $u \in \tau(q)$ and all $q \in Q$. If A is deterministic, then it is called an ℓ-*lr-(R)DFAwtw.

We now study the repetitive NFAwtw and its deterministic counterpart. The following technical result can be derived for the RNFAwtw in the same way as for the NFAwtw.

Lemma 11 *From a given RNFAwtw A, one can construct an RNFAwtw B such that $L(B) = L(A)$ and B only accepts once it has read and deleted its input completely.*

On the other hand, Lemma 2 cannot be extended to the RNFAwtw, if the automaton changes its state at the right sentinel. For example, assume that $\delta(q, \triangleleft) = \{q'\}$, $\delta(q, a) = \emptyset$, and $\tau(q) = \{aa\}$, where q, q' are states of an RDFAwtw A with the input alphabet $\{a\}$. Then, $qaa \cdot \triangleleft \vdash_A q'aa \cdot \triangleleft$, while $qaaa \cdot \triangleleft \vdash_A \mathsf{Reject}$, since $\tau(q) = \{aa\}$ and $\delta(q, a)$ is empty.

The deterministic linear language

$$L_{\text{lin}} = \{ a^n b^n \mid n \geq 1 \}$$

is not accepted by any NFAwtw [16]. Below we shall see that this language is accepted by some repetitive NFAwtw. However, it is still open whether or not this language is accepted by any RDFAwtw.

Lemma 12 *The language $L_{2\text{lin}} = \{ a^{2n}b^{2n} \mid n \geq 1 \}$ is accepted by a repetitive DFAwtw.*

Proof. Let $A_{2\text{lin}} = (Q, \{a, b\}, \triangleleft, \tau, q_0, \delta)$, where $Q = \{q_0, q_1, \ldots, q_8\}$, be the RDFAwtw that is described in Figure 1. Here, in each node, the associated set of translucent words $\tau(q)$ is written under the name of the state $q \in Q$, and there is an oriented edge from a state q to a state q' that is labeled with a letter $x \in \{a, b, \triangleleft\}$, if $\delta(q, x) = q'$. Of course, δ is undefined for all other pairs from $Q \times \{a, b\}$. It can be checked that $A_{2\text{lin}}$ accepts the language $L_{2\text{lin}}$. □

In essentially the same way, also the following result can be proved. A corresponding automaton is presented in Figure 2.

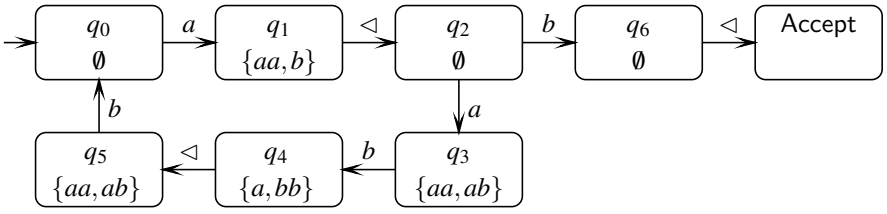

Figure 2: The RDFAwtw $A_{2\text{lin1}}$ for the language $L_{2\text{lin1}} = \{\, a^{2n+1}b^{2n+1} \mid n \geq 0 \,\}$.

Lemma 13 *The language $L_{2\text{lin1}} = \{\, a^{2n+1}b^{2n+1} \mid n \geq 0 \,\}$ is accepted by a repetitive DFAwtw.*

By forming the disjoint union of the RDFAwtws $A_{2\text{lin}}$ and $A_{2\text{lin1}}$, we obtain an RNFAwtw for the language L_{lin}, that is, we have the following consequence.

Corollary 14 *The language $L_{\text{lin}} = \{\, a^n b^n \mid n \geq 1 \,\}$ is accepted by a repetitive NFAwtw.*

Clearly, the language $L_{2\text{lin}}$ does not contain a regular sublanguage that is letter-equivalent to the language itself, as, for each $n \geq 1$, $a^{2n}b^{2n}$ is the only word in $L_{2\text{lin}}$ that has length $4n$. Hence, by Proposition 5, the language $L_{2\text{lin}}$ is not accepted by any NFAwtw. In particular, this shows that Proposition 5 does not extend to the repetitive NFAwtw.

As stated in Corollary 6, each language accepted by an NFAwtw is necessarily semi-linear. This is no longer true if we consider NFAwtws that are repetitive.

Theorem 15 *There exists an RDFAwtw A_{ex} over a binary alphabet such that the language $L(A_{\text{ex}})$ is not semi-linear.*

Proof. We define the RDFAwtw A_{ex} as $A_{\text{ex}} = (Q, \Sigma, \lhd, \tau, q_0, \delta)$, where

$$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_f\},\ \Sigma = \{a, b\},$$

and the functions τ and δ are specified as follows:

$$\begin{array}{llllllllllll}
\tau(q_0) & = & \{ab\}, & \tau(q_1) & = & \emptyset, & \tau(q_2) & = & \{bab\}, & \tau(q_3) & = & \emptyset, \\
\tau(q_4) & = & \{ab\}, & \tau(q_5) & = & \{ab\}, & \tau(q_6) & = & \emptyset, & \tau(q_7) & = & \emptyset, \\
\tau(q_f) & = & \emptyset, & & & & & & & & & \\
\delta(q_0, \lhd) & = & q_1, & \delta(q_1, a) & = & q_2, & \delta(q_2, a) & = & q_2, & \delta(q_2, \lhd) & = & q_3, \\
\delta(q_3, b) & = & q_4, & \delta(q_4, b) & = & q_5, & \delta(q_4, \lhd) & = & q_6, & \delta(q_5, b) & = & q_5, \\
\delta(q_5, \lhd) & = & q_1, & \delta(q_6, a) & = & q_7, & \delta(q_7, b) & = & q_f, & \delta(q_f, \lhd) & = & \mathsf{Accept}.
\end{array}$$

It can now be checked that $L(A_{\text{ex}}) = \{\, (ab)^{2^n} \mid n \geq 1 \,\} = L_{\text{ex}}$. For proving this result, we first establish the following technical statements.

Claim 1. $abab = (ab)^{2^1} \in L(A_{\text{ex}})$.

Proof. Given the word $abab$ as input, the automaton A_{ex} executes the following computation:

$$\begin{array}{llllllll}
q_0abab\cdot\lhd & \vdash_{A_{\text{ex}}} & q_1abab\cdot\lhd & \vdash_{A_{\text{ex}}} & q_2bab\cdot\lhd & \vdash_{A_{\text{ex}}} & q_3bab\cdot\lhd \\
 & \vdash_{A_{\text{ex}}} & q_4ab\cdot\lhd & \vdash_{A_{\text{ex}}} & q_6ab\cdot\lhd & \vdash_{A_{\text{ex}}} & q_7b\cdot\lhd \\
 & \vdash_{A_{\text{ex}}} & q_f\cdot\lhd & \vdash_{A_{\text{ex}}} & \mathsf{Accept}. & &
\end{array}$$

□

Claim 2. For all $n \geq 2$, $q_1(abab)^n \cdot \lhd \vdash_{A_{\rm ex}}^* q_3(bab)^n \vdash_{A_{\rm ex}}^* q_1(ab)^n \cdot \lhd$.

Proof. We proceed by induction on n. If $n = 2$, then we obtain the following computation:

$$\begin{array}{lllllll} q_1(abab)^2 \cdot \lhd & = & q_1ababbabab \cdot \lhd & \vdash_{A_{\rm ex}} & q_2bababbab \cdot \lhd & \vdash_{A_{\rm ex}} & q_2babbbab \cdot \lhd \\ & \vdash_{A_{\rm ex}} & q_3babbbab \cdot \lhd & \vdash_{A_{\rm ex}} & q_4abbbab \cdot \lhd & \vdash_{A_{\rm ex}} & q_5abab \cdot \lhd \\ & \vdash_{A_{\rm ex}} & q_1abab \cdot \lhd & = & q_1(ab)^2 \cdot \lhd. & & \end{array}$$

For the general case, we consider the input $(abab)^{n+1} = abab(abab)^n$:

$$\begin{array}{lllll} q_1abab(abab)^n \cdot \lhd & \vdash_{A_{\rm ex}} & q_2bab(abab)^n \cdot \lhd & \vdash_{A_{\rm ex}} & q_2babbbab(abab)^{n-1} \cdot \lhd \\ & \vdash_{A_{\rm ex}}^{n-1} & q_2bab(bab)^n \cdot \lhd & \vdash_{A_{\rm ex}} & q_3(bab)^{n+1} \cdot \lhd \\ & \vdash_{A_{\rm ex}} & q_4ab(bab)^n \cdot \lhd & \vdash_{A_{\rm ex}} & q_5abab(bab)^{n-1} \cdot \lhd \\ & \vdash_{A_{\rm ex}}^{n-1} & q_5ab(ab)^n \cdot \lhd & \vdash_{A_{\rm ex}} & q_1(ab)^{n+1} \cdot \lhd. \end{array}$$

□

Together Claims 1 and 2 imply that $L_{\rm ex} \subseteq L(A_{\rm ex})$, since, for each $n \geq 2$,

$$q_0(ab)^{2^n} \cdot \lhd \vdash_{A_{\rm ex}} q_1(abab)^{2^{n-1}} \cdot \lhd \vdash_{A_{\rm ex}}^* q_1abab \cdot \lhd \vdash_{A_{\rm ex}}^* \mathsf{Accept}.$$

Conversely, assume that $w \in L(A_{\rm ex})$. Then the computation of the automaton $A_{\rm ex}$ on the input w is accepting, that is, it has the following form:

$$q_0w \cdot \lhd \vdash_{A_{\rm ex}} p_1w_1 \cdot \lhd \vdash_{A_{\rm ex}} p_2w_2 \cdot \lhd \vdash_{A_{\rm ex}} \cdots \vdash_{A_{\rm ex}} p_tw_t \cdot \lhd \vdash_{A_{\rm ex}} \mathsf{Accept},$$

where $t \geq 1$, $p_1, p_2, \ldots, p_t \in Q$, and $w_1, w_2, \ldots, w_t \in \Sigma^*$. From the definition of the functions τ and δ, we see that $p_1 = q_1$ and that $w = (ab)^m$ for some $m \geq 0$, $p_t = q_f$, and $w_t = \lambda$. In fact, as

$$q_0 \cdot \lhd \vdash_{A_{\rm ex}} q_1 \cdot \lhd \vdash_{A_{\rm ex}} \mathsf{Reject}$$

and

$$q_0ab \cdot \lhd \vdash_{A_{\rm ex}} q_1ab \cdot \lhd \vdash_{A_{\rm ex}} q_2b \cdot \lhd \vdash_{A_{\rm ex}} \mathsf{Reject},$$

we can conclude that $m \geq 2$.

Claim 3. For all $n \geq 1$, $(ab)^{2n+1} \notin L(A_{\rm ex})$.

Proof. For the input $(ab)^{2n+1}$, $A_{\rm ex}$ executes the following computation:

$$\begin{array}{lllll} q_0ab(ab)^{2n} \cdot \lhd & \vdash_{A_{\rm ex}} & q_1ab(ab)^{2n} \cdot \lhd & \vdash_{A_{\rm ex}} & q_2bab(ab)^{2n-2}ab \cdot \lhd \\ & \vdash_{A_{\rm ex}}^{n-1} & q_2(bab)^nab \cdot \lhd & \vdash_{A_{\rm ex}} & q_2(bab)^nb \cdot \lhd \qquad \vdash_{A_{\rm ex}} \mathsf{Reject}, \end{array}$$

that is, $A_{\rm ex}$ rejects all uneven powers of ab. □

Thus, it follows that m is an even number. Finally, assume that m is not a power of two, that is, $m = 2^k \cdot r$ for some $k \geq 1$ and an uneven number r. Then, by Claims 2 and 3,

$$q_0(ab)^m \cdot \lhd \vdash_{A_{\rm ex}} q_1(ab)^m \cdot \lhd = q_1(ab)^{2^k \cdot r} \cdot \lhd \vdash_{A_{\rm ex}}^* q_1(ab)^r \cdot \lhd \vdash_{A_{\rm ex}}^* \mathsf{Reject}.$$

In summary, we have shown that $m = 2^n$ for some integer $n \geq 1$, that is, $w = (ab)^{2^n}$ is indeed an element of the language $L_{\rm ex}$. It follows that $L(A_{\rm ex}) = L_{\rm ex}$, which completes the proof of Theorem 15. □

As the language $L_{\rm ex}$ is not semi-linear, this gives the following result.

Corollary 16 *The language class $\mathscr{L}(\mathsf{RDFAwtw})$ contains languages that are not semi-linear.*

As the NFAwtws only accept semi-linear languages, this also implies the following proper inclusions.

Corollary 17 *$\mathscr{L}(\mathsf{DFAwtw}) \subsetneq \mathscr{L}(\mathsf{RDFAwtw})$ and $\mathscr{L}(\mathsf{NFAwtw}) \subsetneq \mathscr{L}(\mathsf{RNFAwtw})$.*

4 Separating the RDFAwtw from the RNFAwtw

The rational trace language

$$L_\vee = \{\, w \in \{a,b\}^* \mid \exists n \geq 0 : |w|_a = n \text{ and } |w|_b \in \{n, 2n\} \,\}$$

is accepted by an NFAwtl, but according to Theorem 8, it is not accepted by any DFAwtw. This means, in particular, that this language separates the DFAwtw from the NFAwtw (see Corollary 9). Here we prove that the language $L_\vee$ is not even accepted by any RDFAwtw.

Theorem 18 $L_\vee \notin \mathscr{L}(\mathsf{RDFAwtw})$.

Proof. Assume to the contrary that there is an RDFAwtw $A = (Q, \Sigma, \lhd, \tau, q_0, \delta)$ on $\Sigma = \{a,b\}$ such that $L(A) = L_\vee$, and let

$$\ell = \max\{\, |u| \mid \exists q \in Q : u \in \tau(q) \,\} \text{ and } k = \max\{\, |\tau(q)| \mid q \in Q \,\},$$

that is, A is ℓ-length-restricted and k-cardinality-restricted. Let $\Lambda > \ell$ be an integer that is sufficiently large. In the following, we consider the accepting computations of A for all inputs of the form $a^n b^n$ and $a^n b^{2n}$, where $n \geq \Lambda$.

As A is repetitive, it may have (one or more) states q for which the set of letters $\Sigma_q^{(A)}$ is empty. In fact, by introducing some additional states with this property, if necessary, we can assume, without loss of generality, that, for each $n \geq \Lambda$, the accepting computation of A on input $w_n = a^n b^n \in L_\vee$ has the following form:

$$\begin{array}{rcccccccccccc} q_0 w_n \cdot \lhd & = & q_0 a^n b^n \cdot \lhd & \vdash_A & p_0 a^n b^n \cdot \lhd & \vdash_A & q_1 z_1 \cdot \lhd & \vdash_A & p_1 z_1 \cdot \lhd & \vdash_A & q_2 z_2 \cdot \lhd \\ & \vdash_A & p_2 z_2 \cdot \lhd & \vdash_A & \cdots & \vdash_A & q_t z_t \cdot \lhd & \vdash_A & p_t z_t \cdot \lhd & \vdash_A & \mathsf{Accept}, \end{array}$$

where, for all $i = 0,1,2,\ldots,t$, $q_i, p_i \in Q$, $\Sigma_{q_i}^{(A)} = \emptyset$, $\delta(q_i, \lhd) = p_i$, $z_i \in \Sigma^{2n-i}$ is obtained from w_n by reading and deleting i letters, $z_t \in (\tau(p_t))^*$, and $\delta(p_t, \lhd) = \mathsf{Accept}$. In addition, $a^n b^n \in (\tau(q_0))^*$ and $z_i \in (\tau(q_i))^*$ for all $i = 1,2,\ldots,t$.

As the set $\tau(q_0)$ is a finite prefix code and $a^n b^n \in (\tau(q_0))^*$, we see that

$$\tau(q_0) \cap (a^* \cdot b^*) = \{a^{i_0}, b^{j_0}\} \cup \{a^{r_1} b^{s_1}, a^{r_2} b^{s_2}, \ldots, a^{r_\nu} b^{s_\nu}\}$$

for some $1 \leq i_0, j_0 \leq \ell$, $\nu \geq 0$, $1 \leq r_1 < r_2 < \cdots < r_\nu < i_0$, and $s_1, s_2, \ldots, s_\nu \geq 1$ such that $r_\mu + s_\mu \leq \ell$ for all $\mu = 1,2,\ldots,\nu$.

Since A is deterministic, $a^m b^m \in L_\vee$, and $a^m b^{2m} \in L_\vee$, we can conclude that $a^m b^m \in (\tau(q_0))^*$ and $a^m b^{2m} \in (\tau(q_0))^*$ for each $m \geq \Lambda$. Hence, for each $m \geq \Lambda$, there exist an index $f_m \in \{1,2,\ldots,\nu\}$ and integers $g_m, h_m \geq 0$ such that $m = g_m \cdot i_0 + r_{f_m} = h_m \cdot j_0 + s_{f_m}$. As $1 \leq r_1 < r_2 < \cdots < r_\nu < i_0$, it follows that $r_{f_m} \equiv m \bmod i_0$ and the index f_m is uniquely determined by i_0 and m. Analogously, it follows that $2m = h'_m \cdot j_0 + s_{f_m}$ for some integer h'_m, which implies that

$$m = 2m - m = h'_m \cdot j_0 + s_{f_m} - (h_m \cdot j_0 + s_{f_m}) = (h'_m - h_m) \cdot j_0.$$

Hence, each sufficiently large integer m is necessarily a multiple of j_0, which means that $j_0 = 1$. Moreover, as $m = g_m \cdot i_0 + r_{f_m}$, either $i_0 = 1$ and $\nu = 0$, or $i_0 > 1$, $\nu = i_0 - 1$, and $r_i = i$ for $i = 1,2,\ldots,\nu$.

If $\delta(p_0,a) = q_1$, then $z_1 = a^{n-1}b^n$ is obtained from $w_n = a^nb^n$ by simply reading and deleting the very first letter. Accordingly, we obtain

$$q_0a^mb^m \cdot \lhd \vdash_A^2 q_1a^{m-1}b^m \cdot \lhd \text{ and } q_0a^mb^{2m} \cdot \lhd \vdash_A^2 q_1a^{m-1}b^{2m} \cdot \lhd$$

for all sufficiently large m. As $\Sigma_{q_1}^{(A)} = \emptyset$ and $\delta(q_1, \lhd) = p_1$, we have

$$q_1a^{m-1}b^m \cdot \lhd \vdash_A p_1a^{m-1}b^m \cdot \lhd \text{ and } q_1a^{m-1}b^{2m} \cdot \lhd \vdash_A p_1a^{m-1}b^{2m} \cdot \lhd$$

for all sufficiently large m. Hence, we can conclude, as above, that

$$\tau(q_1) \cap (a^* \cdot b^*) = \{a^{i_1}, ab^{s'_1}, a^2b^{s'_2}, \ldots, a^{i_1-1}b^{s'_{i_1-1}}, b\}$$

for some $i_1 \geq 1$ and $s'_1, s'_2, \ldots, s'_{i_1-1} \geq 1$.

If $\delta(p_0,a)$ is undefined and $\delta(p_0,b) = q_1$, then $z_1 = a^nb^{n-1}$ is obtained from $w_n = a^nb^n$ by reading and deleting an occurrence of the letter b, that is, a prefix of the form a^nb^i of a^nb^n is in the set $(\tau(p_0))^*$. Again, as $a^mb^m, a^mb^{2m} \in L_\vee$, we see that

$$p_0a^mb^m \cdot \lhd \vdash_A q_1a^mb^{m-1} \cdot \lhd \text{ and } p_0a^mb^{2m} \cdot \lhd \vdash_A q_1a^mb^{2m-1} \cdot \lhd$$

for all sufficiently large m. Hence, we can conclude that

$$\tau(p_0) \cap (a^* \cdot b^*) = \{a^{i_1}, ab^{s'_1}, a^2b^{s'_2}, \ldots, a^{i_1-1}b^{s'_{i_1-1}}\}$$

for some $i_1 \geq 1$ and $s'_1, s'_2, \ldots, s'_{i_1-1} \geq 1$. As $\Sigma_{q_1}^{(A)} = \emptyset$ and $\delta(q_1, \lhd) = p_1$, we have

$$q_1a^mb^{m-1} \cdot \lhd \vdash_A p_1a^mb^{m-1} \cdot \lhd \text{ and } q_1a^mb^{2m-1} \cdot \lhd \vdash_A p_1a^mb^{2m-1} \cdot \lhd$$

for all sufficiently large m, which implies that

$$\tau(q_1) \cap (a^* \cdot b^*) = \{a^{i_2}, ab^{s''_1}, a^2b^{s''_2}, \ldots, a^{i_2-1}b^{s''_{i_2-1}}, b\}$$

for some $i_2 \geq 1$ and $s''_1, s''_2, \ldots, s''_{i_2-1} \geq 1$.

It follows that, for all sufficiently large values of m, the accepting computations of A on input a^mb^m and on input a^mb^{2m} consist of the exactly same sequence of transitional steps until the exponent of one of the factors becomes small.

Now consider a value of n such that, for all $m \geq n$, the common initial part of all the accepting computations of A on input a^mb^m and on input a^mb^{2m} is of length $K > 2 \cdot |Q|$. Then there are indices $0 \leq \alpha < \beta \leq |Q|$ such that the states p_α and p_β are identical. Hence, for all $m \geq n$, we have the following accepting computations:

$$q_0a^mb^m \cdot \lhd \;\vdash_A^{2\cdot\alpha+1}\; p_\alpha z_\alpha \cdot \lhd \;\vdash_A^{2\cdot(\beta-\alpha)}\; p_\beta z_\beta \cdot \lhd \;=\; p_\alpha z_\beta \cdot \lhd \;\vdash_A^*\; \mathsf{Accept}$$

and

$$q_0a^mb^{2m} \cdot \lhd \;\vdash_A^{2\cdot\alpha+1}\; p_\alpha z'_\alpha \cdot \lhd \;\vdash_A^{2\cdot(\beta-\alpha)}\; p_\beta z'_\beta \cdot \lhd \;=\; p_\alpha z'_\beta \cdot \lhd \;\vdash_A^*\; \mathsf{Accept},$$

where z_α is obtained from a^mb^m by reading and deleting α letters, z_β is obtained from z_α by reading and deleting further $\beta - \alpha$ letters, z'_α is obtained from a^mb^{2m} by reading and deleting α letters, and z'_β is obtained from z'_α by reading and deleting further $\beta - \alpha$ letters. Thus,

$$z_\alpha = a^{m-i_\alpha}b^{m-j_\alpha},\ z_\beta = a^{m-i_\alpha-i_\beta}b^{m-j_\alpha-j_\beta},\ z'_\alpha = a^{m-i_\alpha}b^{2m-j_\alpha},\ z'_\beta = a^{m-i_\alpha-i_\beta}b^{2m-j_\alpha-j_\beta}$$

for some integers $i_\alpha + j_\alpha = \alpha$ and $i_\beta + j_\beta = \beta - \alpha$.

Consider now the input $a^{m+i_\beta} b^{m+j_\beta}$. Then

$$\begin{array}{lcllcl} q_0 a^{m+i_\beta} b^{m+j_\beta} \cdot \lhd & \vdash_A^{2\cdot\alpha+1} & p_\alpha a^{m+i_\beta-i_\alpha} b^{m+j_\beta-j_\alpha} \cdot \lhd & \vdash_A^{2\cdot(\beta-\alpha)} & p_\beta a^{m-i_\alpha} b^{m-j_\alpha} \cdot \lhd \\ & = & p_\alpha a^{m-i_\alpha} b^{m-j_\alpha} \cdot \lhd & = & p_\alpha z_\alpha \cdot \lhd \\ & \vdash_A^* & \mathsf{Accept}. & & \end{array}$$

As m is large and $i_\beta, j_\beta \leq \beta < |Q|$, we see that $m + j_\beta < 2m < 2(m+i_\beta)$. Hence, $a^{m+i_\beta} b^{m+j_\beta} \in L(A) = L_\vee$ implies that $i_\beta = j_\beta$. However, we also have the following computation:

$$\begin{array}{lcllcl} q_0 a^{m+i_\beta} b^{2m+j_\beta} \cdot \lhd & \vdash_A^{2\cdot\alpha+1} & p_\alpha a^{m+i_\beta-i_\alpha} b^{2m+j_\beta-j_\alpha} \cdot \lhd & \vdash_A^{2\cdot(\beta-\alpha)} & p_\beta a^{m-i_\alpha} b^{2m-j_\alpha} \cdot \lhd \\ & = & p_\alpha a^{m-i_\alpha} b^{2m-j_\alpha} \cdot \lhd & = & p_\alpha z'_\alpha \cdot \lhd \\ & \vdash_A^* & \mathsf{Accept}. & & \end{array}$$

Now $m + i_\beta < 2m + j_\beta = 2m + i_\beta < 2m + 2i_\beta = 2 \cdot (m + i_\beta)$ implies that $a^{m+i_\beta} b^{2m+j_\beta} \notin L_\vee$, a contradiction. This proves that the language $L_\vee$ is not accepted by an RDFAwtw. □

Hence, we have the following separation result.

Corollary 19 $\mathscr{L}(\mathsf{RDFAwtw}) \subsetneq \mathscr{L}(\mathsf{RNFAwtw})$.

5 Emptiness Is Undecidable for RDFAwtws

From an NFAwtw A, an NFA B can be constructed such that $L(B)$ is a sublanguage of $L(A)$ that is letter-equivalent to $L(A)$ (see Proposition 5). As the emptiness problem is decidable for NFAs (even in polynomial time), and as $L(A)$ is empty if and only if $L(B)$ is empty, it thus follows that the emptiness problem is decidable for NFAwtws. In contrast to this fact, we now prove that this problem is undecidable for repetitive DFAwtws. Our proof exploits a reduction from the *Post Correspondence Problem* (PCP), which can be stated as follows (see, e.g., [4]):

Instance : Two non-erasing morphisms $f, g : \Sigma^* \to \Delta^*$.
Question : Is there a non-empty word $w \in \Sigma^+$ such that $f(w) = g(w)$?

It is well-known that the PCP is undecidable in general, even when it is restricted to a binary alphabet Δ.

Theorem 20 *The emptiness problem is undecidable for RDFAwtws.*

Proof. Let $\Sigma = \{x_1, x_2, \ldots, x_m\}$ for some $m \geq 2$, let $\Delta = \{a, b\}$, where we can assume without loss of generality that the two alphabets Σ and Δ are disjoint, and let $f, g : \Sigma^* \to \Delta^*$ be two non-erasing morphisms, that is, $f(x_i) = u_i$ and $g(x_i) = v_i$ are non-empty words over Δ for all $1 \leq i \leq m$.

In addition, let $\Delta' = \{a', b'\}$ be a new alphabet such that Δ' is disjoint from Σ and Δ, and let $\varphi' : \Delta^* \to \Delta'^*$ be the morphism induced by mapping a to a' and b to b'. Finally, let $\Omega = \Sigma \cup \Delta \cup \Delta'$, let $\pi_a : \Omega^* \to \Delta^*$ be the projection from Ω^* onto Δ^*, and let $\pi' : \Omega^* \to \Delta^*$ be the morphism that is defined through $\pi'(x_i) = \lambda$ for all $1 \leq i \leq m$, $\pi'(a) = \pi'(b) = \lambda$, and $\pi'(a') = a$ and $\pi'(b') = b$. Then $\varphi' \circ \pi'$ is the projection from Ω^* onto Δ'^*.

We now define an RDFAwtw $A_{(f,g)} = (Q, \Omega, \lhd, \tau, q_0, \delta)$ by taking

$$Q = \{q_0, q_1, q_2\} \cup \bigcup_{i=1}^{m} \left(\{ p_y^{(i)} \mid y \text{ is a proper prefix of } u_i \} \cup \{ q_y^{(i)} \mid y \text{ is a proper prefix of } v_i \} \right)$$

and by defining the functions τ and δ as follows, where $\mathrm{pref}(u_i, j)$ denotes the prefix of u_i of length j, and $\mathrm{pref}(v_i, j)$ denotes the prefix of v_i of length j:

$$
\begin{array}{lcl}
\tau(q_0) & = & \Sigma \cup \{aa', bb'\}, \\
\tau(q_1) & = & \emptyset, \\
\tau(q_2) & = & \emptyset, \\
\tau(p_y^{(i)}) & = & \Sigma \cup \{a', b'\} \text{ for all } p_y^{(i)} \in Q, \\
\tau(q_y^{(i)}) & = & \Sigma \cup \{a, b\} \text{ for all } q_y^{(i)} \in Q,
\end{array}
$$

$$
\begin{array}{lcl}
\delta(q_0, \lhd) & = & q_1, \\
\delta(q_1, x_i) & = & p_\lambda^{(i)} \text{ for } 1 \leq i \leq m, \\
\delta(p^{(i)}_{\mathrm{pref}(u_i,j)}, a) & = & p^{(i)}_{\mathrm{pref}(u_i,j+1)} \text{ for } 1 \leq i \leq m \text{ and } 0 \leq j < |u_i| - 1, \text{ if } \mathrm{pref}(u_i, j+1) = \mathrm{pref}(u_i, j)a, \\
\delta(p^{(i)}_{\mathrm{pref}(u_i,j)}, b) & = & p^{(i)}_{\mathrm{pref}(u_i,j+1)} \text{ for } 1 \leq i \leq m \text{ and } 0 \leq j < |u_i| - 1, \text{ if } \mathrm{pref}(u_i, j+1) = \mathrm{pref}(u_i, j)b, \\
\delta(p^{(i)}_{\mathrm{pref}(u_i,|u_i|-1)}, a) & = & q_\lambda^{(i)} \text{ for } 1 \leq i \leq m, \text{ if } u_i = \mathrm{pref}(u_i, |u_i| - 1)a, \\
\delta(p^{(i)}_{\mathrm{pref}(u_i,|u_i|-1)}, b) & = & q_\lambda^{(i)} \text{ for } 1 \leq i \leq m, \text{ if } u_i = \mathrm{pref}(u_i, |u_i| - 1)b,
\end{array}
$$

$$
\begin{array}{lcl}
\delta(q^{(i)}_{\mathrm{pref}(v_i,j)}, a') & = & q^{(i)}_{\mathrm{pref}(v_i,j+1)} \text{ for } 1 \leq i \leq m \text{ and } 0 \leq j < |v_i| - 1, \text{ if } \mathrm{pref}(v_i, j+1) = \mathrm{pref}(v_i, j)a, \\
\delta(q^{(i)}_{\mathrm{pref}(v_i,j)}, b') & = & q^{(i)}_{\mathrm{pref}(v_i,j+1)} \text{ for } 1 \leq i \leq m \text{ and } 0 \leq j < |v_i| - 1, \text{ if } \mathrm{pref}(v_i, j+1) = \mathrm{pref}(v_i, j)b, \\
\delta(q^{(i)}_{\mathrm{pref}(v_i,|v_i|-1)}, a') & = & q_2 \text{ for } 1 \leq i \leq m, \text{ if } v_i = \mathrm{pref}(v_i, |v_i| - 1)a, \\
\delta(q^{(i)}_{\mathrm{pref}(v_i,|v_i|-1)}, b') & = & q_2 \text{ for } 1 \leq i \leq m, \text{ if } v_i = \mathrm{pref}(v_i, |v_i| - 1)b, \\
\delta(q_2, x_i) & = & p_\lambda^{(i)} \text{ for all } 1 \leq i \leq m, \\
\delta(q_2, \lhd) & = & \mathsf{Accept},
\end{array}
$$

and δ is undefined for all other pairs from $Q \times \Omega$.

It can now be verified that the language $L(A_{(f,g)})$ is non-empty if and only if the instance (f,g) of the PCP has a solution. In fact, let $\psi_2 : \Delta^* \to (\Delta \cup \Delta')^*$ be the morphism that is defined through $a \mapsto aa'$ and $b \mapsto bb'$. It can be checked that the language $L(A_{(f,g)})$ contains some words from the shuffle of $x_{i_1}x_{i_2}\cdots x_{i_r}$ and $\psi_2(f(x_{i_1}x_{i_2}\cdots x_{i_r}))$ for each solution $x_{i_1}x_{i_2}\cdots x_{i_r}$ of (f,g). □

If $x = x_{i_1}x_{i_2}\cdots x_{i_r}$ is a solution of the PCP instance (f,g), also x^n is a solution of (f,g) for each $n \geq 2$. Hence, it follows that the language $L(A_{(f,g)})$ is either empty or infinite, and it is infinite if and only if (f,g) has a solution. This has the following consequence.

Corollary 21 *The finiteness problem is undecidable for RDFAwtws.*

An RDFAwtw for the empty language is easily obtained. Accordingly, the undecidability of the emptiness problem implies the following undecidability results.

Corollary 22 *The inclusion problem and the equivalence problem are undecidable for RDFAwtws.*

Let (f,g) be an instance of the PCP, and let $A_{(f,g)}$ be the resulting RDFAwtw as constructed in the proof of Theorem 20. Assume that the language $L(A_{(f,g)})$ is regular. Then also the language

$$L_{(f,g)} = L(A_{(f,g)}) \cap (\Sigma^* \cdot \{aa', bb'\}^*)$$

is regular. It can be checked that $L_{(f,g)}$ consists of all words of the form $w\psi_2(w)$, where $w \in \Sigma^+$ is a solution for the instance (f,g) of the PCP.

Assume that (f,g) admits a solution $w \in \Sigma^+$. Then, for all $n \geq 2$, also w^n is a solution for (f,g), that is, $w^n(\psi_2(w))^n \in L_{(f,g)}$. Let k be the number of states of a minimal DFA for the language $L_{(f,g)}$. Now pumping arguments show that, for all $n > k$, there exists an integer μ, $1 \leq \mu < k$, such that $w^{n+\mu}(\psi_2(w))^n$ is an element of the language $L_{(f,g)}$. However, this contradicts the above observation about the form of the elements of this set, as $w \neq \lambda$. It follows that the set $L_{(f,g)}$, and therewith the language $L(A_{(f,g)})$, is not regular whenever (f,g) has a solution. As the empty set is regular, this yields the following undecidability result.

Corollary 23 *The regularity problem is undecidable for RDFAwtws.*

Finally, a language $L \subseteq \Gamma^*$ is called *bounded* if there exist finitely many non-empty words $w_1, w_2, \ldots, w_k \in \Gamma^*$ such that L is contained in the regular language $w_1^* \cdot w_2^* \cdots w_k^*$. Now the boundedness problem is the problem of deciding whether a given language L is bounded. A recent survey on the status of this problem for various types of automata can be found in [5]. While it is still open whether or not the boundedness problem is decidable for DFAwtws, we have the following undecidability result.

Corollary 24 *The boundedness problem is undecidable for RDFAwtws.*

Proof. Let (f,g) be an instance of the PCP and let $A_{(f,g)}$ be the RDFAwtw obtained from (f,g) as in the proof of Theorem 20. We now modify this RDFAwtw as follows.

Let $\Gamma = \{c,d\}$ be a new alphabet that is disjoint from Ω, let $\Omega' = \Omega \cup \Gamma$, let q_3 be a new state, and let the functions τ and δ be modified as follows:

$$\tau'(q) = \left\{ \begin{array}{ll} \emptyset, & \text{if } q = q_3 \\ \Sigma \cup \{aa', bb'\} \cup \Gamma, & \text{if } q = q_0, \\ \tau(q), & \text{otherwise} \end{array} \right\} \text{ and } \delta'(q,x) = \left\{ \begin{array}{ll} q_3, & \text{if } q = q_2 \text{ and } x \in \Gamma \cup \{\lhd\}, \\ q_3, & \text{if } q = q_3 \text{ and } x \in \Gamma, \\ \mathsf{Accept}, & \text{if } q = q_3 \text{ and } x = \lhd, \\ \delta(q,x), & \text{otherwise.} \end{array} \right.$$

Let $A'_{(f,g)}$ be the new RDFAwtw. If (f,g) does not have a solution, then $L(A'_{(f,g)})$ is empty, and hence, it is bounded. However, if (f,g) has a solution $w \in \Sigma^+$, then $L(A'_{(f,g)})$ contains all words of the form $w\psi_2(w)z$, where $z \in \Gamma^*$, which shows that this language is not bounded. Thus, $L(A'_{(f,g)})$ is bounded if and only if (f,g) does not have a solution. As $A'_{(f,g)}$ is easily constructed from (f,g), this yields the undecidability of the boundedness problem. □

6 Conclusion

We have shown that, by adding the property of repetitiveness, the expressive capacity of the finite automata with translucent words is indeed severely extended. However, the following topics remain to be studied:

1. The closure properties for the classes $\mathscr{L}(\mathsf{RDFAwtw})$ and $\mathscr{L}(\mathsf{RNFAwtw})$: It is easily seen that $\mathscr{L}(\mathsf{RNFAwtw})$ is closed under union. On the other hand, the results on the language $L_\vee$ imply that the class $\mathscr{L}(\mathsf{RDFAwtw})$ is neither closed under union nor under alphabetic morphisms. Moreover, by using the same proof idea as for NFAwtws, it can be shown that the complement of a language that is accepted by an RDFAwtw is accepted by an RNFAwtw. However, it remains open whether or not this deterministic class is closed under complementation. Finally, it is still open whether or not this class is closed under intersection (with regular languages). Obviously, it is closed under intersection with sets of the form K^*, where K is a finite prefix code.
2. What can we say about the complexity of the membership problem for an RDFAwtw? Obviously, this problem is decidable in quadratic time, but can we do better than that?

References

[1] Simon Beier & Markus Holzer (2022): *Nondeterministic right one-way jumping finite automata*. *Information and Computation* 284, p. 104687, doi:10.1016/j.ic.2021.104687.

[2] Suna Bensch, Henning Bordihn, Markus Holzer & Martin Kutrib (2009): *On input-revolving deterministic and nondeterministic finite automata*. *Information and Computation* 207, pp. 1140–1155, doi:10.1016/j.ic.2009.03.002.

[3] Hiroyuki Chigahara, Szilárd Zsolt Fazekas & Akihiro Yamamura (2016): *One-way jumping finite automata*. *International Journal of Foundations of Computer Science* 27, pp. 391–405, doi:10.1142/S0129054116400165.

[4] Tero Harju & Juhani Karhumäki (1997): *Morphisms*. In Grzegorz Rozenberg & Arto Salomaa, editors: *Handbook of Formal Languages*, 1, Springer, Berlin, pp. 439–510, doi:10.1007/978-3-642-59136-5_7.

[5] Oscar H. Ibarra & Ian McQuillan (2024): *Techniques for showing the decidability of the boundedness problem of language acceptors*. In Joel D. Day & Florin Manea, editors: *DLT 2024, Proc.*, Lecture Notes in Computer Science 14791, Springer, Cham, Switzerland, pp. 156–172, doi:10.1007/978-3-031-66159-4_12.

[6] František Mráz & Friedrich Otto (2022): *Non-returning finite automata with translucent letters*. In Henning Bordihn, Gésa Horváth & György Vaszil, editors: *12th International Workshop on Non-Classical Models of Automata and Applications (NCMA 2022)*, *EPTCS* 367, pp. 143–159, doi:10.4204/EPTCS.367.10.

[7] František Mráz & Friedrich Otto (2023): *Non-returning deterministic and nondeterministic finite automata with translucent letters*. *RAIRO Theoretical Informatics and Applications* 57, p. 34, doi:10.1051/ita/2023009.

[8] František Mráz & Friedrich Otto (2024): *Repetitive finite automata with translucent letters*. In Florin Manea & Giovanni Pighizzini, editors: *14th International Workshop on Non-Classical Models of Automata and Applications (NCMA 2024), Proc.*, *EPTCS* 407, pp. 150–167, doi:10.4204/EPTCS.407.11.

[9] František Mráz & Friedrich Otto (2025): *On a measure for the descriptional complexity of finite automata with translucent words*. In Andreas Malcher & Luca Prigioniero, editors: *DCFS 2025, Proc.*, *Lecture Notes in Computer Science* 15759, Springer, Cham, Switzerland, pp. 180–195, doi:10.1007/978-3-031-97100-6_13.

[10] Benedek Nagy & László Kovács (2014): *Finite automata with translucent letters applied in natural and formal language theory*. In Ngoc Thanh Nguyen, Ryszard Kowalczyk, Ana Fred & Filipe Joaquim, editors: *Transactions on Computational Collective Intelligence XVII*, *Lecture Notes in Computer Science* 8790, Springer, Heidelberg, pp. 107–127, doi:10.1007/978-3-662-44994-3_6.

[11] Benedek Nagy & Friedrich Otto (2010): *CD-systems of stateless deterministic R(1)-automata accept all rational trace languages*. In Adrian-Horia Dediu, Henning Fernau & Carlos Martín-Vide, editors: *LATA 2010, Proc.*, *Lecture Notes in Computer Science* 6031, Springer, Berlin, pp. 463–474, doi:10.1007/978-3-642-13089-2_39.

[12] Benedek Nagy & Friedrich Otto (2011): *Finite-state acceptors with translucent letters*. In Gemma Bel-Enguix, Veronica Dahl & Alfonso O. De La Puente, editors: *BILC 2011: AI Methods for Interdisciplinary Research in Language and Biology, Proc.*, SciTePress, Portugal, pp. 3–13, doi:10.5220/0003272500030013.

[13] Benedek Nagy & Friedrich Otto (2011): *Globally deterministic CD-systems of stateless R(1)-automata*. In Adrian-Horia Dediu, Shunsuke Inenaga & Carlos Martín-Vide, editors: *Language and Automata Theory and Applications, LATA 2011, Proc.*, *Lecture Notes in Computer Science* 6638, Springer, Berlin, pp. 390–401, doi:10.1007/978-3-642-21254-3_31.

[14] Benedek Nagy & Friedrich Otto (2012): *On CD-systems of stateless deterministic R-automata with window size one*. *Journal of Computer and System Sciences* 78, pp. 780–806, doi:10.1016/j.jcss.2011.12.009.

[15] Benedek Nagy & Friedrich Otto (2013): *Globally deterministic CD-systems of stateless R-automata with window size 1*. *International Journal of Computer Mathematics* 90, pp. 1254–1277, doi:10.1080/00207160.2012.688820.

[16] Benedek Nagy & Friedrich Otto (2024): *Finite automata with sets of translucent words*. In Joel D. Day & Florin Manea, editors: *DLT 2024, Proc.*, Lecture Notes in Computer Science 14791, Springer, Cham, Switzerland, pp. 236–251, doi:10.1007/978-3-031-66159-4_17.

[17] Benedek Nagy & Friedrich Otto (2024): *A two-dimensional infinite hierarchy for finite automata with translucent words*. Submitted.

[18] Friedrich Otto (2023): *A survey on automata with translucent letters*. In Benedek Nagy, editor: *CIAA 2023, Proc.*, Lecture Notes in Computer Science 14151, Springer, Cham, Switzerland, pp. 21–50, doi:10.1007/978-3-031-40247-0_2.

[19] Friedrich Otto (2025): *Restarting Automata – Extensions and Generalizations*. Theory and Applications of Computability, Springer, Cham, Switzerland, doi:10.1007/978-3-031-78701-0.

A Myhill-Nerode Type Characterization of 2detLIN Languages

Benedek Nagy
Department of Mathematics, Eastern Mediterranean University
99628 Famagusta, North Cyprus, Mersin-10, Turkey
Department of Computer Science, Institute of Mathematics and Informatics,
Eszterházy Károly Catholic University, Eger, Hungary
nbenedek.inf@gmail.com

Linear automata are automata with two reading heads starting from the two extremes of the input, are equivalent to $5' \to 3'$ Watson-Crick (WK) finite automata. The heads read the input in opposite directions and the computation finishes when the heads meet. These automata accept the class LIN of linear languages. The deterministic counterpart of these models, on the one hand, is less expressive, as only a proper subset of LIN, the class 2detLIN is accepted; and on the other hand, they are also equivalent in the sense of the class of the accepted languages. Now, based on these automata models, we characterize the class of 2detLIN languages with a Myhill-Nerode type of equivalence classes. However, as these automata may do the computation of both the prefix and the suffix of the input, we use prefix-suffix pairs in our classes. Additionally, it is proven that finitely many classes in the characterization match with the 2detLIN languages, but we have some constraints on the used prefix-suffix pairs, i.e., the characterization should have the property to be complete and it must not have any crossing pairs.

1 Introduction

In formal language theory, the class of regular languages plays a crucial role, similar as finite automata in automata theory. They are widely applied and there are several theoretical studies known about them. One important fact is the characterization of regular languages by the Myhill-Nerode theorem [15, 33]. In a nutshell, every regular language induces finitely many equivalence classes of words considering them as possible prefixes of the words of the language. This "if and only if" characterization, in fact, gives also the minimal completely defined deterministic finite automaton for each regular language and thus, it has very important practical consequences. The number of states of such minimal automaton is the same as the number of equivalence classes above, for each language. This measure is the most known and most used measure for descriptional complexity of regular languages. There are other known measures, e.g., transition complexity [6], nondeterministic state and transition complexities [37], union-complexity [20, 23], just to mention a few.

In this paper, we consider a proper superclass of the class of regular languages based on a kind of deterministic 2-head automata. This model starts the computation by having its two heads at the two extremes of the input: the first head may read the first and the second head may read the last letter of the input. The computation goes step by step till the heads meet (at some position of the input). If the automaton is in a final state at that time, then the computation is accepting and the input is in the accepted language. The class of the nondeterministic variant of these automata accepts the class of linear languages, another well-known class of formal languages. It is properly between the regular and context-free classes. Here, usually, we refer to this model of automata as linear automata (based on [14, 28]).

N. Moreira and L. Prigioniero (Eds.): 15th International Workshop on Non-Classical Models of Automata and Applications (NCMA 2025).
EPTCS 422, 2025, pp. 73–88, doi:10.4204/EPTCS.422.6

However, very similar models were defined also under various names, e.g., 2-head automata [18] or bi-automata [10].
We also recall the concept of Watson-Crick finite automata which belongs to a special field of DNA computing. From the end of the last century, DNA computing has emerged as a relatively new computational paradigm [35]. Watson-Crick automata (abbreviated as WK automata) have been introduced in [5], for details and early results see also [35]. A WK automaton works on a double-stranded tape called Watson-Crick tape (i.e., on a DNA molecule), whose strands are scanned separately by read only heads. The symbols in the corresponding cells of the double-stranded tapes are related by the Watson-Crick complementarity relation (a symmetric and bijective relation in the nature with pairs Adenine-Thymine and Cytosine-Guanine). The two strands of a DNA molecule have opposite $5' \to 3'$ orientations. The $5' \to 3'$ WK automata are more realistic in the sense that both heads use the same biochemical direction (that is, actually, opposite physical directions) [12, 13, 16]. A WK automaton is sensing if it knows whether the heads are at the same position. The sensing $5' \to 3'$ WK finite automata work essentially in the same way as linear automata, but they may read strings in a transition. Their 1-limited variant, in which exactly one letter is read in each transition, has the same power, i.e., they accept the same family of languages as the original model ([17, 30, 31]). There are numerous variants of these automata where some extensions or restrictions are applied including stateless [25], state- and quasi-deterministic and reversible variants [22, 24, 32], jumping $5' \to 3'$ WK automata [11], as well as, $5' \to 3'$ WK multi-counter and pushdown automata [3, 4, 7, 21] and $5' \to 3'$ WK automata accepting necklaces [26], just to mention a few.

We are interested in a proper subclass of the linear languages, namely 2detLIN, the class that is accepted by the deterministic variant of the linear automata (and of the sensing $5' \to 3'$ WK automata), as they are described in details in, e.g., [19, 29]. This class is still a proper superset of the class of regular languages. Here, we give a characterization of 2detLIN that is somewhat similar to Myhill-Nerode characterization of the regular languages. It is done by using prefix-suffix pairs. We show some important properties of the pairs that can be used in the characterization. Although there are significant differences between the original Myhill-Nerode characterization result and our result, we believe that our results could lead to a kind of similar descriptional complexity measure to a larger class of languages than the original results which can be used for the class of regular languages.

Because of the page limit some of the proofs are omitted.

2 Definitions and Preliminaries

We assume that the reader is familiar with the basic concepts of formal languages and automata, otherwise she or he is referred, e.g., to [8, 36] for the concepts not explained in detail here. We denote the empty word by λ. The set of nonnegative integers is denoted by $\mathbb{N}$.

There are various classes in the Chomsky hierarchy. We briefly recall here the classes of regular and linear languages. A *generative grammar* is a four tuple (N,T,S,P) with two disjoint, finite, nonempty alphabets N and T, where the former is called nonterminal alphabet, the latter is called terminal alphabet. The symbol $S \in N$ is the start (a.k.a. sentence) symbol and P is the finite set of productions (a.k.a. rewriting rules). Each production is of the form $u \to v$ where u must contain at least one nonterminal symbol. A generative grammar (N,T,S,P) is *regular* (in some places they are also called right-linear) if each production of the grammar is in one of the following forms: $A \to w$ (with $A \in N, w \in T^*$) and $A \to wB$ (with $A,B \in N, w \in T^*$). Further, a generative grammar is *linear* if each of its productions is in one of the following forms: $A \to w$ (with $A \in N, w \in T^*$) and $A \to uBv$ (with $A,B \in N, u,v \in T^*$). Obviously, every regular grammar is also linear at the same time. These classes of grammars generate

the classes of regular and linear languages, respectively. We recall here some special linear grammars: if in a linear grammar for each production with a nonterminal on the right side $A \to uBv$, $|u| = n$, $|v| = m$ holds, then the grammar is called *k-rated linear* with the value $k = \frac{m}{n}$, $(m,n \in \mathbb{N}, n \neq 0)$ [2, 9]. These grammars generate *k-rated linear languages*. The union of the sets of k-rated linear languages for any nonnegative rational value of k is called the family of *fix-rated linear languages*. Observe that, in fact, the 0-rated linear grammars and languages are the regular grammars and languages. The 1-rated linear grammars and languages are usually referred to as *even-linear grammars and languages* ([1, 39]).

The classes of regular and linear languages can be accepted by the class of traditional finite automata and a class of 2-head automata, respectively. Let us discuss, first, the case of regular languages. Now, let us recall the concept of finite automata. A five tuple $A = (Q,T,q_0,\delta,F)$ is a *finite automaton* with the finite nonempty set of states Q, with a finite nonempty input alphabet T, an initial state $q_0 \in Q$, a set of final (a.k.a. accepting) states F and a transition function δ. The latter is defined, in general, as $\delta : Q \times (T \cup \{\lambda\}) \to 2^Q$. In this general case, the model is known as nondeterministic finite automata. There is a more restricted version of the finite automata with $\delta : Q \times T \to Q$ called *deterministic finite automata*. Automata are used to accept formal languages. It is well-known that the classes of both the nondeterministic and deterministic finite automata recognize exactly the class of regular languages. There are some 2-head extensions of these traditional models, that play a central role for us. A five tuple $A' = (Q,T,q_0,\delta,F)$ is a 2-head finite automaton (a.k.a. *linear automaton*, [14, 28]) where Q,T,q_0 and F have the same roles as in traditional finite automata, but δ is defined in a different way: $\delta : (Q \times T \times \{\lambda\} \cup Q \times \{\lambda\} \times T) \to 2^Q$.

Further, a configuration of a linear automaton is a pair (q,w) where q is the current state of the automaton and w is the part of the input word which has not been processed (read) yet. For $w' \in T^*$, $xy \in T$, $q,q' \in Q$, we write a computation step between two configurations as: $(q,xw'y) \Rightarrow (q',w')$ if and only if $q' \in \delta(q,x,y)$. Notice that in such a computation step either $x \in T$ and $y = \lambda$ or $y \in T$ and $x = \lambda$, i.e., exactly one of the heads is reading an input letter. We denote the reflexive and transitive closure of the relation $\Rightarrow$ (one step of a computation) by $\Rightarrow^*$, and refer to it as the computation relation. Therefore, for a given $w \in T^*$, an accepting computation is a sequence of computation steps of the form $(q_0,w) \Rightarrow^* (q,\lambda)$, starting from the initial state and ending in a state $q \in F$ with no input left. Finally, the language accepted by a linear automaton M is:

$$L(M) = \{w \in T^* | (q_0,w) \Rightarrow^* (q,\lambda), q \in F\}.$$

Note that here we have defined a kind of restricted (1-limited) variant, where exactly one input letter is read in each transition. (In the more general variant both heads may read a letter in a transition, however, the described model is equivalent to this more general model in the sense of the class of the accepted languages.)

It is known that the class of linear automata accepts the class of linear languages. Now, we are interested in the deterministic variant of them. As usual, we say that an automaton is deterministic if at any possible configuration there is at most one way to continue the computation. The deterministic counterpart of linear automata can accept only a special subclass of the class of linear languages, the *class 2detLIN*. It is known that this class is a superclass of the class of regular languages containing various interesting linear languages, including all fix-rated linear languages. On the other hand, 2detLIN is incomparable to the class detLIN, the class accepted by deterministic one-turn pushdown automaton (with the deterministic counterpart of another well-known automata model accepting the class of linear languages).

We also recall that a closely related model, the sensing $5' \to 3'$ Watson-Crick finite automata work in a very similar manner. There is a very important difference between the 2-head automata model we

have defined and the Watson-Crick automata, namely that the latter models are able to read strings in a transition. Nevertheless, in [31] and in [29] it is proven that this feature does not help for the model to accept larger classes of languages than the classes LIN and 2detLIN, respectively. Therefore, we may use the definition we gave above to define the class we are interested in.

Further, we may assume that all states of the automaton A is reachable, i.e., for each state q, there is an input word w_q such that the computation of w_q ends in state q: $(q_0, w_q) \Rightarrow^* (q, \lambda)$. (Those states that are not reachable do not have any effect on the computations of the automaton, and thus, they can simply be removed form the set of states without changing the accepted language.) This assumption could be important when some properties of the automaton are analyzed, e.g., in the proposition below.

Formally, we can write that a linear automaton is deterministic, if and only if for each pair of $w \in T^*$ and $q \in Q$ there exists at most one $w' \in T^*$ and $q' \in Q$ such that $(q, w) \Rightarrow (q', w')$. This property is defined as a constraint on all possible computations of the automaton, however, it gives restriction for the used automaton itself. Thus, deterministic linear automata can be characterized as follows.

Proposition 1 *A linear automaton is deterministic if and only if for each q of its states, either*

- $\delta(q, a, \lambda) = \emptyset$ *for all* $a \in T$*; and* $|\delta(q, \lambda, a)| \leq 1$ *for each* $a \in T$*;*
  or
- $\delta(q, \lambda, a) = \emptyset$ *for all* $a \in T$*; and* $|\delta(q, a, \lambda)| \leq 1$ *for each* $a \in T$.

We refer to the transitions $\delta(q, a, b) = \emptyset$ ($ab \in T$, i.e., one of a and b is a letter, the other is λ), as transitions which are not defined in the automaton. Thus, we may interpret the previous statement as follows. In a deterministic linear automaton at each state, we may have transitions defined only for at most one of the heads.

In automata theory, there are usually two main variants of the used deterministic finite automata. If a finite automaton is incomplete (we say this, when its transition function is only a partial function), it may happen that the automaton is unable to read (and thus to accept) some of the possible input words, in these cases, the automaton gets stuck and the computation halts without accepting. In contrast, in the case of a completely defined finite automaton, the automaton can read any input and can do the computation such that the whole input has been processed. Somewhat similarly, we may also define this variant of linear automata. The main difference in the work of the "incomplete" and "completely defined" (shortly, complete) linear automata is the same as at finite automata, however, based on Proposition 1, we may characterize the latter ones as follows.

Proposition 2 *A deterministic linear automaton is complete if and only if for each q of its states, either*

- $\delta(q, a, \lambda) = \emptyset$ *for all* $a \in T$*; and* $\delta(q, \lambda, a) \in Q$ *for each* $a \in T$*;*
  or
- $\delta(q, \lambda, a) = \emptyset$ *for all* $a \in T$*; and* $\delta(q, a, \lambda) \in Q$ *for each* $a \in T$.

Moreover, for each deterministic linear automaton, we may construct a complete deterministic linear automaton accepting the same language by adding a sink state, if necessary. This technique is similar to the one used in the case of deterministic finite automata for the regular languages.

Based on the previous proposition and fact we may always assume that our linear automaton accepting a language in 2detLIN is complete.

About the work of linear deterministic automata we state the following useful property. It is a kind of analogous property of the deterministic finite automata that when it does the computation on an input w, then the initial part of the computation is the same as the computation on a prefix of w. As linear automata may consume the input from its both extremes, we have a somewhat more complex statement and therefore we state it formally.

Lemma 1 *Let a complete deterministic linear automaton A and an input word $w \in T^*$ be given. Let the computation on w by A be $(q_0,w) \Rightarrow^* (q,\lambda)$ such that, the prefix u and the suffix v of w ($w = uv$) were read by the first and second head, respectively, during the computation. This computation is a $|w|$-step long computation. Then for any input $uw'v$ with $w' \in T^*$, the first $|w|$ steps of the computation are $(q_0,uw'v) \Rightarrow^* (q,w')$.*

We note here that in [27] for similar models, specific functions were defined and used that give the following information for every input: which of the heads is stepping in which step of the computation and which head reads the given letter of the input.

Finally, in this section we recall a very important and useful characterization of the regular languages.

Let a language $L \subseteq T^*$ be given. Based on it, we define the equivalence relation: for any $x,y \in T^*$,

$$x \equiv_L y \text{ if and only if } xw \in L \Leftrightarrow yw \in L \text{ for every } w \in T^*.$$

That is, two words are equivalent if exactly the same continuations of them are in L. The number of the equivalence classes of the relation $\equiv_L$ is called the index of the language L. By the Myhill-Nerode theorem, a language L is regular if and only if the relation $\equiv_L$ has a finite index, i.e., the number of the equivalence classes is finite. Moreover, the index of a regular language is then the same as the minimal number of the states in a completely defined finite automaton accepting the language L.

In this paper, our aim is to give a kind of similar if and only if characterization of the languages in the class 2detLIN.

3 Equivalent classes by pairs of prefixes and suffixes

As the computation on the input by linear automata goes by reading not only the prefix, but maybe also the suffix of the input word, we use *pre*fix-*su*ffix-pairs (shortly, *presu*s) in our characterization. Let us consider a language L over the alphabet T. We say that the prefix-suffix-pair (u_1,v_1) is equivalent to the presu (u_2,v_2) with respect to the language L, if for every word $w \in T^*$, $u_1wv_1 \in L \Leftrightarrow u_2wv_2 \in L$. We call a set of equivalence classes of presus a *border classification*, BC for short. However, a BC not need to cover all prefix-suffix pairs. We also define *pseudo BC*s, in which in each class, the presus are equivalent to each other, but it may happen that some of the classes contain presus that are also equivalent to each other. From a pseudo BC, a BC can be obtained by joining those classes that contain presus that are equivalent to each other. We say that a (pseudo) BC contains a presu (u,v), if it appears in a class of the (pseudo) BC.

To characterize the languages of 2detLIN, we need some additional conditions. In the sequel, we list them.

Definition 1 *We say that a BC (or a pseudo BC) is* complete*, if for each word $w \in T^*$ it contains exactly one pair (u,v) such that $w = uv$.*

Definition 2 *We say that a BC (or a pseudo BC) has a* crossing pair*, if it contains both presus (u_1,v_1), (u_2,v_2) where u_1 is a proper prefix of u_2 and v_2 is a proper suffix of v_1. The presus (u_1,v_1), (u_2,v_2) are referred as a crossing pair.*

For better understanding these concepts we show some examples.

Example 1 *Let us consider the regular language a^*b^*. One may consider BC Ω_1 with only one class containing all pairs of the form (a^*,λ). It is easy to see that Ω_1 is not complete, since, for instance, there is no presu (u,v) in it with $uv = aaab$. On the other hand, Ω_1 does not contain any crossing pairs.*

Consider now, the BC Ω_2 with three classes $C_1 = \{(u,v) \mid u \in a^*, v \in b^*\}$ and $C_2 = \{(u,v) \mid u \in a^*b^*b, v \in b^*\}$ and $C_3 = \{(u,v) \mid u \in a^*, v \in aa^*b^*\}$. The BC Ω_2 is not complete, since, e.g., for the word ab it contains the presus $(a,b) \in C_1$ and $(ab,\lambda) \in C_2$. Furthermore, Ω_2 contains crossing pairs, as $(aaa,b) \in C_1$ and $(a,abb) \in C_3$ appear in it.

Definition 3 *Let us fix a language L and a BC for L. The index of the BC is the number of equivalence classes in it.*

The following statement is a direct consequence of the definitions.

Lemma 2 *Let a pseudo BC Ω be given for a language L. Then, there is a BC for L that has index at most the number of classes in the pseudo BC Ω.*
Further, in general, if for a language L there is a pseudo BC with finitely many classes, then there is a BC for L with a finite index.

We need the following technical lemma that describes an important behaviour of our automata.

Lemma 3 *By any complete deterministic linear automaton A, every word w is processed in a unique way and thus, there is exactly one presu (u,v) with $w = uv$ such that A reads u by the first head and v by the second head when performing the computation on the input w.*

Now, we are ready to state and prove one of our main results, the characterization of 2detLIN languages by finitely many equivalence classes of presus.

Theorem 1 *A language L is in 2detLIN if and only if there is a complete BC with a finite index for L that does not contain any crossing pairs.*

Proof The proof is constructive in both directions. First, let us prove that for each language L in 2detLIN, there is a complete BC with finite index as it is stated.

Let $A = (Q,T,q_0,\delta,F)$ be a completely defined deterministic linear automaton accepting L with the set of states $Q = \{q_0,\ldots,q_n\}$. Based on A, we construct a complete pseudo BC. Basically, the construction follows Algorithm 1 that is described below.

Algorithm 1.
Input: $A = (\{q_0,\ldots,q_n\},T,q_0,\delta,F)$, a complete deterministic linear automaton.
Output: a pseudo BC for the language accepted by A.
Put (λ,λ) representing the empty word into class C_0.
Let the set J of states initially contain only q_0 and let the set J' be empty.
While (True) do
  For each i in $\{0,\ldots,n\}$ do
    If $(q_i \in J)$
      For each $a \in \Sigma$ do
        If $(\delta(q_i,a,\lambda) = q_j)$
          Put q_j into the set J'
          For each $(u,v) \in C_i$ do
            Put the presu $((ua,v))$ into C_j
        If $(\delta(q_i,\lambda,a) = q_j)$
          Put q_j into the set J'
          For each $(u,v) \in C_i$ do
            Put the presu $((u,av))$ into C_j
  Let $J = J'$ and J' be empty.

Note that as we have infinitely many presus, the algorithm is running for the infinity, however, it puts the presus in the appropriate classes by their increasing values of the sum of the lengths of prefix and suffix in a pair. The algorithm works in a somewhat similar manner as a breadth-first search algorithm build an infinite tree level by level. Thus, for each presu it will be clear after a finitely many steps where it belongs if it appears in the pseudo BC (as we claim it later).

Clearly the set J contains always a subset of Q. It is clear that in the beginning this subset contains only q_0. In each iteration of the while loop the new presus appear in the classes that have their sum of the length of prefix and suffix that is one more as similar values of the presus in the previous iteration. For us, at this moment, the only important is that we can decide which pair appears in the constructed pseudo BC. Moreover, if it appears in it, then we can also decide in which class it is. See, Claim 1.

Claim 1. The classes obtained by Algorithm 1 form a pseudo BC for the language L accepted by the given complete deterministic linear automaton A.

Further, for each presu (u,v) it is clear if it appears in the created pseudo BC, and if so, it is clear where, into which class C_j it belongs. Moreover, the induced pseudo BC is complete.

By continuing the proof of the theorem, it is already clear that the induced pseudo BC is complete. What is left to be shown is that this pseudo BC does not contain any crossing pairs.

Claim 2. For any complete deterministic linear automaton A, the obtained pseudo BC does not contain any crossing pairs.

By the construction, as we have seen, we obtained a pseudo BC, if two presus are in the same class C_i then they must be equivalent. We have proven that there are finitely many classes C_i; further the contained presus imply a complete pseudo BC without crossing pairs. This, by Lemma 2 also proves that there is a complete BC with finite index without crossing pairs, since by joining some classes of the pseudo BC, its completeness and crossing-freeness properties are not changing. Thus, the first part of the proof has been finished.

(We note here that in a pseudo BC some of the sets C_i may contain presus that are equivalent to each other. This property is somewhat similar that a deterministic finite automaton that is not minimal has some states that represent prefix words belonging to the same Myhill-Nerode class.)

Now, we prove the other direction. Thus, let us assume that for a language L, a complete BC Ω is given without crossing pairs, then we define a deterministic linear automaton that accepts L (matching with Ω), and thus the language that is characterized by Ω is a 2detLIN language. Thus, let finitely many equivalence classes $C_1,\ldots,C_n$ of presus be given in Ω, our aim is to construct a deterministic linear automaton $A=(Q,T,q_0,\delta,F)$ based on that. As the BC is defined for a language, the alphabet T is fixed, and it will be used for A. Further, we assign two states q_i and p_i for each class C_i. As the given BC is complete, it contains a presu representing the empty word, and it must be (λ,λ). Let the initial state be one of the states that represents the class C_i which contains (λ,λ). However, to know which of those, first, we need some technical arguments.

Since no crossing pairs occur in the BC Ω and it is complete, we can deduce the following statements.

Claim 3. Let a complete BC for a language be given without crossing pairs. If $(u,v)\in C_i$ corresponds to the word uv in the BC, then for each $a\in T$ either (ua,v) or (u,av) corresponds to uav.

When $(u,v)\in C_i$ corresponds to the word uv in a complete BC, then we may also say that (u,v) represents the word uv.

Claim 4. Let a complete BC for a language be given without crossing pairs. If (u,v) is in the BC, then either (ua,v) is in the BC for all $a\in T$ or (u,av) is in the BC for all $a\in T$.

Based on Claims 3 and 4, we can now continue our construction. Applying Claim 4, for the pair $(\lambda,\lambda) \in C_i$, either (a,λ) is in Ω or (λ,a). In the former case, let q_i be the initial state; in the latter case, let p_i be the initial state. (This is independent of which element $a \in T$ is considered).

Generally, the equivalence classes of the BC Ω are partitioned into two sets, one containing all presus (u,v) such that (ua,v) is in some C_j in Ω, while the other one contains the presu (u,v) for (u,av) is in some C_j. We label the first mentioned set by the state q_i for the equivalence class C_i and the second one by the state p_i. We define the transition function of the automaton A such that A ends up in state q_i or p_i, respectively, if it reads a word uv for which (u,v) is in the according equivalence class. Thus, for each presu (u,v) and for each letter $a \in T$, let us consider the word uav. As the BC is complete, it appears in the BC represented by exactly one presu, and either the left or the right head reads the last letter a between u and v, i.e., either (ua,v) or (u,av) appears in the BC, respectively. However, it may happen that there are two equivalent presus (u_1,v_1) and (u_2,v_2) in a class C_j such that for (u_1,v_1) the first, but for (u_2,v_2) the second head will make the next read (we show such example later). Therefore, automaton A will be in state q_j after processing words represented by the first type presus, and in p_j after processing words represented by the second type presus.

Formally, for each state q_i and for each letter $a \in T$, we define either

- the transition $\delta(q_i,a,\lambda) = q_j$ if there is a presu $(u,v) \in C_i$ such that $(ua,v) \in C_j$ and (uaa,v) appears in the BC; or
- $\delta(q_i,a,\lambda) = p_j$ if there is a presu $(u,v) \in C_i$ such that $(ua,v) \in C_j$ and (ua,av) appears in the BC.

Further, for each state p_i and for each letter $a \in T$, we define either

- the transition $\delta(p_i,\lambda,a) = q_j$ if $(u,v) \in C_i$ and $(u,av) \in C_j$ and (ua,av) appears in Ω; or
- the transition $\delta(p_i,\lambda,a) = p_j$ if $(u,v) \in C_i$ and $(u,av) \in C_j$ and (u,aav) appears in Ω.

Clearly, for each state and letter, exactly one of the above transitions will be defined for A based on the properties shown in the previous Claims.

Thus, we can deduce that the transition function determines a complete deterministic linear automaton.

Only one thing is left to define: the set of accepting states F. This is based, actually, not on the BC itself, but on some property used to define Ω. In Ω, the equivalence classes are defined based on how the possible middle part of the input (i.e., the part we put between the prefix and suffix of the presu) behaves, i.e., with which middle part the input will belong to the language. Now, let $F = \{q_i, p_i \mid C_i \text{ contains presus } (u,v) \text{ such that } uv \in L\}$.

Based on the construction, it can be seen that A accepts the language L. •

By the first half of the proof, we are sure that the number of classes in a BC for a 2detLIN language L is not more than the number of states of a complete deterministic linear automaton that accepts L. However, we have seen (by the other direction of the proof) that there could be a BC such that it may require a larger (at most twice much) number of states in an accepting linear automaton.

We show some examples. Our first example is very characteristic: the languages of palindromes are in 2detLIN (for any alphabet), but not deterministic linear as for alphabets which are at least binary, there is no deterministic one-turn pushdown automata accepting them. In fact these languages are 1-rated, i.e., even linear.

Example 2 *Let us consider the alphabet $T = \{a,b,c\}$. The table of an automaton A that accepts the language of palindromes (the language containing a word w if and only if its reversal w^R is the same as itself) over T is given below in a form of a Cayley table:*

$T \setminus Q$	q_0 *(left)*	q_1 *(right)*	q_2 *(right)*	q_3 *(right)*	q_4 *(left)*
a	q_1	q_0	q_4	q_4	q_4
b	q_2	q_4	q_0	q_4	q_4
c	q_3	q_4	q_4	q_0	q_4

Further q_0 is the initial state, and q_0, q_1, q_2, q_3 are the accepting states, while q_4 is, in fact, the sink state. After the name of each state, it is indicated which of the heads can move in transitions from that state.

The equivalence classes of presus based on this automaton are:

- C_0: $\{(u, u^R) \mid u \in T^*\}$,
- C_1: $\{(ua, u^R) \mid u \in T^*\}$,
- C_2: $\{(ub, u^R) \mid u \in T^*\}$,
- C_3: $\{(uc, u^R) \mid u \in T^*\}$,
- C_4: $\{(uev, fu^R) \mid u, v \in T^*, e, f \in T, e \neq f\}$.

Clearly $(\lambda, \lambda) \in C_0$ and, by applying Algorithm 1, the set J contains only C_0. Since the first head can read in q_0, the pairs (a, λ), (b, λ) and (c, λ) are created and they are put to classes C_1, C_2 and C_3, respectively. Then, the new set J contain C_1, C_2 and C_3. In the next iteration, taking C_1 first, some new presus appear in the BC: $(a, a) \in C_0$, $(a, b), (a, c) \in C_4$ and both C_0 and C_4 are appended to J'. Then C_2 and C_3 are considered in a similar manner to put some new presus into some classes. Then, updating the set J of states, a new iteration comes. It can be seen that following the algorithm, the above classes are obtained.

Our next example, is a non-regular, fix-rated linear language over the binary alphabet which can be accepted both by deterministic linear automata and deterministic one-turn pushdown automata.

Example 3 *Now, let us consider the language $L = \{1^n 0^{3n} \mid n \in \mathbb{N}\}$, this language is in fact a 3-rated linear language and it is both in detLIN and 2detLIN.*

Let us consider the following BC for L (on the left).

- C_1: $\{(1^n, 0^{3n}) \mid n \in \mathbb{N}\}$,
- C_2: $\{(1^{n+1}, 0^{3n}) \mid n \in \mathbb{N}\}$,
- C_3: $\{(1^{n+1}, 0^{3n+1}) \mid n \in \mathbb{N}\}$,
- C_4: $\{(1^{n+1}, 0^{3n+2}) \mid n \in \mathbb{N}\}$,
- C_5: $\{(1^n, 0^{3n+1}) \mid n \in \mathbb{N}\}$,
- C_6: $\{(0w, \lambda) \mid w \in \{0,1\}^*\} \cup \{(1w, 1) \mid w \in \{0,1\}^*\} \cup \{(1^{m+1}w, 10^{3m+1}) \mid m \in \mathbb{N}, w \in \{0,1\}^*\} \cup \{(1^{m+1}w, 10^{3m+2}) \mid m \in \mathbb{N}, w \in \{0,1\}^*\} \cup \{(1^m w, 10^{3m}) \mid m \in \mathbb{N}, m > 0, w \in \{0,1\}^*\} \cup \{(1^m 0w, 0^{3m+1}) \mid m \in \mathbb{N}, m > 0, w \in \{0,1\}^*\}$.

It is easy to see that the sets C_i are pairwise disjoint, moreover, the BC is complete as it contains a pair for every word (actually, C_6 guarantees this fact). Based on that we may have the complete deterministic linear automaton A accepting L (see the table below).

$T \setminus Q$	q_1	q_2	q_3	q_4	q_5	q_6	p_1	p_2	p_3	p_4	p_5	p_6
0	q_6	$-$	$-$	$-$	q_6	q_6	q_5	p_3	p_4	p_1	$-$	$-$
1	p_2	$-$	$-$	$-$	p_3	q_6	q_6	q_6	q_6	q_6	$-$	$-$

For each state q_i the first, for each state p_i the second head can read the input in the next step. Further, as $(\lambda,\lambda) \in C_1$ and, e.g., $(1,\lambda)$ in the BC, q_1 is the initial state.
The final states are q_1, p_1 as only class C_1 contains presus representing words of L. Observe that, in fact, the states q_2, q_3, q_4, p_5, p_6 are not reachable from q_0, thus one may simply erase them from the automaton. Thus, in fact the obtained linear automaton has 7 states (it is complete and deterministic). Observe that class C_6 contains the presus that cannot be continued by inserting a word to the middle to get a word of language L. Some of the words belonging to these presus are clearly representing something outside of the language, as for instance every word starting with a 0 is in $\{(0w,\lambda) \mid w \in \{0,1\}^*\}$, or every word ending with a 1 is either in the above set or in $\{(1w,1) \mid w \in \{0,1\}^*\}$. On the other hand, the presu $(110,0000000)$ is in the set $\{(1^m0w, 0^{3m+1}) \mid m \in \mathbb{N}, m > 0, w \in \{0,1\}^*\}$, thus it also belongs to C_6 even if it represents the word 110^8, however, "it was read not in a correct way" by the heads, thus no continuation of the computation reading it will be accepting.

Neither the automaton nor the characterization by BC, in the previous example, are the simplest one for L, however, our aim is to show that our theory works also if not the most efficient description is given if it meets the requirements (e.g., finiteness, completeness). Actually, in the example there are both types of presus in class C_1, thus both the states q_1 and p_1 are required to be in the automaton.

In the next example we highlight the property that a complete deterministic linear automaton may have states for the same class of presus with different head movements.

Example 4 *Let the language L of the even-length palindromes over $\{a,b\}$ be considered. The following automaton accepts it:*

$T \setminus Q$	q_1	p_1	p_2	q_3	p_3	p_4	p_5	q_6	q_7	q_8
a	p_2	q_8	q_1	p_4	q_7	q_7	q_7	q_7	q_7	p_1
b	q_3	q_6	q_7	p_5	p_1	p_2	p_3	p_1	q_7	q_7

where the initial state is q_1 and the accepting states are q_1, p_1 and p_5. For each state q_i the first, for each state p_i the second head can read a letter from the input.

The corresponding classes of presus are belonging to the following languages, i.e., for each class C_i, any of the words of L_i can be put into the middle to have a word in L.

1. *C_1 for states q_1 and p_1:* $L_1 = \{w \mid w \text{ is an even-length palindrome}\} = L$.
2. *C_2 for state p_2:* $L_2 = L \cdot \{a\}$.
3. *C_3 for states q_3 and p_3:* $L_3 = L \cdot \{b\}$.
4. *C_4 for state p_4:* $L_4 = L \cdot \{ab\}$.
5. *C_5 for state p_5:* $L_5 = L \cdot \{bb\} \cup \{\lambda\}$.
6. *C_6 for state q_6:* $L_6 = \{b\} \cdot L$.
7. *C_7 for state q_7:* $L_7 = \{\}$, *there is no way to make it acceptable.*
8. *C_8 for state q_8:* $L_8 = \{a\} \cdot L$.

Finally, we may also use our result to show that a language is not in 2detLIN as we present in the next example.

Example 5 *Let us consider the language $L = \{a^n b^n c^n \mid n \in \mathbb{N}\}$. We show that L is not a 2detLIN language by contradiction. Thus, let us assume that we have a complete BC without crossing pairs with a finite index for L. Let the number of equivalence classes be i. Further, let us assume that there is a deterministic linear automaton A that accepts L (based on the BC given above).*

There are words in the language with arbitrarily long prefix from a^* and arbitrarily long suffix from c^*. Thus, let us consider presus in the form (a^m, c^k). We show that not any two different presus in this form can be in the same class. Let (a^m, c^k) and (a^j, c^ℓ) two different presus. Let us use the notation $max_1 = \max\{m,k\}$ and let $w = a^{max_1-m}b^{max_1}c^{max_1-k}$. Since the two presus are not the same at least one of $m \neq j$ and $k \neq \ell$ holds. Then,

- *on the one hand, presu (a^m, c^k) with the word w results $a^m a^{max_1-m}b^{max_1}c^{max_1-k}c^k = a^{max_1}b^{max_1}c^{max_1} \in L$, but*
- *on the other hand, presu (a^j, c^ℓ) with the word w results $a^j a^{max_1-m}b^{max_1}c^{max_1-k}c^\ell = a^{max_1+j-m}b^{max_1}c^{max_1+\ell-k}$. However, in either case, this word is not in L, since in the first case, the number of a-s does not match with the number of b-s, and in the second case, the number of c-s does not match with the number of b-s.*

Considering the word $a^{2i}b^{2i}c^{2i} \in L$, the complete BC must contain at least $2i$ presus of the form (a^m, c^k) that belong to the first $2i$ steps of an accepting computation of this word by the deterministic linear automaton A. However, each of these presus must be in a unique class which contradicts to the fact that there are only i classes.

4 Discussion

Now, let us discuss what can we gain and what we cannot gain by such characterizations. For the regular languages, the Myhill-Nerode characterization is closely related to the minimal deterministic finite automaton accepting the language, as we have recalled. Moreover, as this minimal automaton is unique (up to renaming the states), it also allows to identify a language.

The case of 2detLIN is different, we may have various orders/ways to consume the prefix and the suffix of the input. However, we have some strong analogies. As for the original Myhill-Nerode theorem, an automaton accepting the considered language L may have computations that equivalent words lead the automaton to the same state. Based on the (second half of) the proof of Theorem 1, we state the following analogous result for 2detLIN languages in the form of a theorem.

Theorem 2 *The BC characterization of a 2detLIN language L allows us to have a deterministic linear automaton A accepting L such that there are at most two states for each equivalent set of presus. Moreover, in the computations of any two equivalent presus, after processing these prefix and suffix pairs, A is in one of these two states (let us denote them by q_i and p_i for class C_i). If A has both of them, then in one of them the first, in the other the second head can move. If a presu (u_1, v_1) is in the class C_i, then each input having the prefix-suffix pair u_1, v_1 is processed by A through one of the states q_i or p_i: either $(q_0, u_1wv_1) \Rightarrow^* (q_i, w)$ for all $w \in T^*$ or $(q_0, u_1wv_1) \Rightarrow^* (p_i, w)$ for all $w \in T^*$. If A has both p_i and q_i, then there is also a presu (u_2, v_2) in C_i, such that $(q_0, u_2wv_2) \Rightarrow^* (r, w)$ for all $w \in T^*$, where $r \in \{q_i, p_i\}$, but it differs from the state used for presu (u_1, v_1).*

Let us discuss, now, cases where we may have a similarly powerful characterization as the original Myhill-Nerode result for the regular languages. It is proven in [19] that all k-rated linear languages for all nonnegative rational values of k are in 2detLIN. More precisely, it is shown that the set of fix-rated linear languages is a proper subset of 2detLIN.

Theorem 3 *Let us consider a k-rated linear language L with $k = \frac{m}{n}$ with co-primes m and n. Then L has a complete (pseudo) BC without crossing pairs such that for all presus in the class "always the same head is stepping" in a corresponding automaton. More precisely, if (u_1, v_1) and (u_2, v_2) are both in the*

class C_i, then either both (u_1a, v_1) and (u_2a, v_2) are in the BC, and they are in the same class C_a for each $a \in T$, respectively; or both (u_1, av_1) and (u_2, av_2) are in the BC, and they are in the same class C'_a for each $a \in T$, respectively. Moreover, the corresponding complete deterministic linear automaton reads every input with an alternating usage of the heads as follows:
Till the whole input is processed,

- *it reads a letter by the first head from the left of the input in n computation steps, then*
- *in the next m computation steps, it reads the input by the second head from the right.*

When the last letter is read by a head (depending on the length of the original input), the computation finishes and the acceptance is decided.

We **conjecture** that the minimal automaton (with the parameter k) can be defined and determined such that it has the minimal number of states among the complete deterministic linear automata accepting L and having the above fixed property about the order of the head steps. Further, this minimal automaton can be used as a unique representant of the given k-rated linear language, and thus, also language equality of these languages can be decided in these classes similarly, as by the original Myhill-Nerode theorem language equivalence of regular languages can be decided.

It is important to use co-primes m and n, otherwise the characterization gives a larger number of classes and states. Moreover, the characterization depends on the value of k. As every regular language is k-rated with any positive rational value of k (see, e.g., [9, 38]), this result could give also several alternative characterizations for regular languages.

Corollary 1 *As for a special subclass, for the regular languages as 0-rated linear languages, exactly the original Myhill-Nerode characterization comes as a special case of our main theorem (Theorem 1) with Theorem 3.*

Now, we discuss further properties of BCs and coin various open problems.

As each regular language is k-rated linear for any positive rational k, there is already a large ambiguity to describe them based on Theorem 3 by fixing the value of k in almost arbitrary way. An interesting question could be how we can find a value of k such that the number of classes will be optimal, i.e., maybe less than their number in the original $k = 0$ case. Could it also happen that a minimal representation of a regular language is not connected to any specific value of k, that is, the representation does not consider the language as a fix-rated linear?

Now, on the other hand, when a general 2detLIN language is considered, we know that there is a BC for it that has the finite index property. On the other hand, there could be various complete BCs without crossing pairs with finite indices for the same language. Thus, neither the classes, nor their number, nor the number of states of an accepting complete deterministic linear automaton are uniquely defined. Therefore, to find the minimal value of classes and/or the minimal number of states of a complete deterministic linear automaton accepting the language are also open questions.

Furthermore, since the linear automata have two heads, we already have some kind of ambiguity based on that, i.e., the order in which the heads process the input may vary from one automaton to other accepting the same language. Moreover, if the order of head movements does not fit for the language, one may also find BC with infinite index representing a 2detLIN language. This can be done, e.g., in the way how a non-regular language is characterized by the original Myhill-Nerode classes: If one uses in the BC only pairs, where, let us say, the second element, the suffix is always λ guessing that the language can be processed by a linear automaton where only the first head is used. We get a complete BC without crossing pairs, but since the language is not regular, this BC has an infinite index (similarly as it has

infinite index by using only prefixes). Therefore, it is crucial to find a kind of efficient representation with a BC to prove that the language is in 2detLIN.

Therefore, we may conclude that in general, we may not be able to identify a 2detLIN language by a given BC. More precisely, for the same language there are various BCs, but for a BC, the language is precisely defined if it is also known which of the equivalent classes contain presus (u,v) with the property that $uv \in L$. As we have no bijection between BCs and languages, trying to apply this method for language equivalence in general, may need some further techniques to be involved.

Finally, we show another way how our result is applicable. Note that various closure properties of 2detLIN were established in [19] and in [34, 29].

Proposition 3 *Let L be a 2detLIN language. For L^c, the complement of L, the same partitions, i.e., equivalence classes can be used as for L.*

Proof Let a completely defined linear automaton $A = (Q,T,q_0,\delta,F)$ for L be given. Then, it has the same set Q of states as a completely defined automaton A' accepting L^c, with the same transition function. Only the set of accepting states is complemented, i.e., in A' it is $Q \setminus F$. Thus based on the transition function and on the set of states, the equivalence classes of presus are the same for these two languages. •

5 Conclusions

The class of sensing $5' \to 3'$ Watson-Crick automata, as well as the class of linear automata, accept exactly the linear languages [14, 16, 19, 31]. Their deterministic counterparts accept a class that is a proper superset of the class of regular, but at the same time, it is a proper subset of the class of linear languages. This class is denoted by 2detLIN. Based on deterministic linear automata and on the way they do their computations on the input, we characterized the languages of this class by using equivalent prefix-suffix pairs (abbreviated as presus in the paper, while their partitioning into equivalence classes is abbreviated as BC standing for border classification). We have shown that if there is complete BC for a language L with finitely many equivalence classes without crossing pairs, then the language L is in 2detLIN and vice versa. In this way, by our results, on the one hand, the class 2detLIN can be further analysed using this new type of description. The connection of the number of equivalence classes and the number of states in an accepting minimal complete deterministic linear automaton is not as straightforward as in the case of regular languages. In case of regular languages, the equivalence classes based only on the prefixes are used and their number is the same as the number of states of a minimal completely defined deterministic finite automaton accepting the language. However, we believe that the characterization presented here can be connected to a descriptional complexity measure for 2detLIN, or at least for the class of fixed linear languages, i.e., for a proper superclass of the set of regular languages. The next steps to this direction are left for future research. On the other hand, for some languages we are able also to prove that they are not in the class 2detLIN based on our results.

Acknowledgments

The author is very grateful to the reviewers for their valuable comments.

References

[1] V. Amar & Gianfranco R. Putzolu (1964): *On a Family of Linear Grammars*. *Inf. Control.* 7(3), pp. 283–291, doi:10.1016/S0019-9958(64)90294-3.

[2] V. Amar & Gianfranco R. Putzolu (1965): *Generalizations of Regular Events*. *Inf. Control.* 8(1), pp. 56–63, doi:10.1016/S0019-9958(65)90275-5.

[3] Ömer Egecioglu, László Hegedüs & Benedek Nagy (2010): *Stateless multicounter $5' \to 3'$ Watson-Crick automata*. In: *Fifth International Conference on Bio-Inspired Computing: Theories and Applications, BICTA 2010, University of Hunan, Liverpool Hope University, Liverpool, United Kingdom / Changsha, China, September 8-10 and September 23-26, 2010*, IEEE, pp. 1599–1606, doi:10.1109/BICTA.2010.5645263.

[4] Ömer Egecioglu, László Hegedüs & Benedek Nagy (2011): *Hierarchies of Stateless Multicounter $5' \to 3'$ Watson-Crick Automata Languages*. *Fundam. Informaticae* 110(1-4), pp. 111–123, doi:10.3233/FI-2011-531.

[5] Rudolf Freund, Gheorghe Păun, Grzegorz Rozenberg & Arto Salomaa (1997): *Watson-Crick finite automata*. In Harvey Rubin & David Harlan Wood, editors: *DNA Based Computers, Proceedings of a DIMACS Workshop, Philadelphia, Pennsylvania, USA, June 23-25, 1997*, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 48, DIMACS/AMS, pp. 297–327, doi:10.1090/dimacs/048/22.

[6] Yuan Gao, Kai Salomaa & Sheng Yu (2010): *Transition Complexity of Incomplete DFAs*. In Ian McQuillan & Giovanni Pighizzini, editors: *Proceedings Twelfth Annual Workshop on Descriptional Complexity of Formal Systems, DCFS 2010, Saskatoon, Canada, 8-10th August 2010*, *EPTCS* 31, pp. 99–109, doi:10.4204/EPTCS.31.12.

[7] László Hegedüs, Benedek Nagy & Ömer Egecioglu (2012): *Stateless multicounter $5' \to 3'$ Watson-Crick automata: the deterministic case*. *Nat. Comput.* 11(3), pp. 361–368, doi:10.1007/S11047-011-9290-9.

[8] John E. Hopcroft & Jeffrey D. Ullman (1979): *Introduction to Automata Theory, Languages and Computation*. Addison Wesley. Available at https://api.semanticscholar.org/CorpusID:31901407.

[9] Géza Horváth & Benedek Nagy (2010): *Pumping lemmas for linear and nonlinear context-free languages*. *Acta Univ. Sapientiae Informatica* 2(2), pp. 194–209, doi:10.48550/arXiv.1012.0023. Available at https://acta.sapientia.ro/en/series/informatica/publications/informatica-contents-of-volume-2-number-2-2010/-pumping-lemmas-for-linear-and-nonlinear-context-free-languages.

[10] Ondrej Klíma & Libor Polák (2011): *On Biautomata*. In Rudolf Freund, Markus Holzer, Carlo Mereghetti, Friedrich Otto & Beatrice Palano, editors: *Third Workshop on Non-Classical Models for Automata and Applications - NCMA 2011, Milan, Italy, July 18 - July 19, 2011. Proceedings*, *books@ocg.at* 282, Austrian Computer Society, pp. 153–164.

[11] Radim Kocman, Zbynek Krivka, Alexander Meduna & Benedek Nagy (2022): *A jumping $5' \to 3'$ Watson-Crick finite automata model*. *Acta Informatica* 59(5), pp. 557–584, doi:10.1007/S00236-021-00413-X.

[12] Peter Leupold & Benedek Nagy (2009): *5' → 3' Watson-Crick Automata with Several Runs*. In Henning Bordihn, Rudolf Freund, Markus Holzer, Martin Kutrib & Friedrich Otto, editors: *Workshop on Non-Classical Models for Automata and Applications - NCMA 2009, Wroclaw, Poland, August 31 - September 1, 2009. Proceedings*, *books@ocg.at* 256, Austrian Computer Society, pp. 167–180.

[13] Peter Leupold & Benedek Nagy (2010): *$5' \to 3'$ Watson-Crick Automata With Several Runs*. *Fundam. Informaticae* 104(1-2), pp. 71–91, doi:10.3233/FI-2010-336.

[14] Roussanka Loukanova (2007): *Linear Context Free Languages*. In Cliff B. Jones, Zhiming Liu & Jim Woodcock, editors: *Theoretical Aspects of Computing - ICTAC 2007, 4th International Colloquium, Macau, China, September 26-28, 2007, Proceedings*, *Lecture Notes in Computer Science* 4711, Springer, pp. 351–365, doi:10.1007/978-3-540-75292-9_24.

[15] J. Myhill (1957): *Finite automata and the representation of events*. *WADD* TR-57-624, pp. 112–137.

[16] Benedek Nagy (2008): *On $5' \to 3'$ Sensing Watson-Crick Finite Automata*. In Max H. Garzon & Hao Yan, editors: *DNA Computing, 13th International Meeting on DNA Computing, DNA13, Memphis, TN, USA, June 4-8, 2007, Revised Selected Papers*, *Lecture Notes in Computer Science* 4848, Springer, pp. 256–262, doi:10.1007/978-3-540-77962-9_27.

[17] Benedek Nagy (2009): *On a hierarchy of $5' \to 3'$ sensing WK finite automata languages*. In: *Mathematical Theory and Computational Practice, CiE 2009, Abstract Booklet, Heidelberg, Germany*, pp. 266–275.

[18] Benedek Nagy (2012): *A class of 2-head finite automata for linear languages*. *Triangle* 8, pp. 89–99.

[19] Benedek Nagy (2013): *On a hierarchy of $5' \to 3'$ sensing Watson–Crick finite automata languages*. *Journal of Logic and Computation* 23(4), pp. 855–872, doi:10.1093/logcom/exr049. arXiv:https://academic.oup.com/logcom/article-pdf/23/4/855/2775832/exr049.pdf.

[20] Benedek Nagy (2019): *Union-Freeness, Deterministic Union-Freeness and Union-Complexity*. In Michal Hospodár, Galina Jirásková & Stavros Konstantinidis, editors: *Descriptional Complexity of Formal Systems - 21st IFIP WG 1.02 International Conference, DCFS 2019, Košice, Slovakia, July 17-19, 2019, Proceedings*, *Lecture Notes in Computer Science* 11612, Springer, pp. 46–56, doi:10.1007/978-3-030-23247-4_3.

[21] Benedek Nagy (2020): *$5' \to 3'$ Watson-Crick pushdown automata*. *Inf. Sci.* 537, pp. 452–466, doi:10.1016/J.INS.2020.06.031.

[22] Benedek Nagy (2021): *State-deterministic $5' \to 3'$ Watson-Crick automata*. *Nat. Comput.* 20(4), pp. 725–737, doi:10.1007/S11047-021-09865-Z.

[23] Benedek Nagy (2022): *Operational union-complexity*. *Inf. Comput.* 284, p. 104692, doi:10.1016/J.IC.2021.104692.

[24] Benedek Nagy (2022): *Quasi-deterministic $5' \to 3'$ Watson-Crick Automata*. In Henning Bordihn, Géza Horváth & György Vaszil, editors: *Proceedings 12th International Workshop on Non-Classical Models of Automata and Applications, NCMA 2022, Debrecen, Hungary, August 26-27, 2022*, *EPTCS* 367, pp. 160–176, doi:10.4204/EPTCS.367.11.

[25] Benedek Nagy (2023): *On language classes accepted by stateless $5' \to 3'$ Watson-Crick finite automata*. *Annales Mathematicae et Informaticae* 58, pp. 110–120, doi:10.33039/ami.2023.08.004.

[26] Benedek Nagy (2024): *$5' \to 3'$ Watson-Crick Automata accepting Necklaces*. In Florin Manea & Giovanni Pighizzini, editors: *Proceedings 14th International Workshop on Non-Classical Models of Automata and Applications (NCMA 2024), NCMA 2024, Göttingen, Germany, 12-13 August 2024*, *EPTCS* 407, pp. 168–185, doi:10.4204/EPTCS.407.12.

[27] Benedek Nagy & Zita Kovács (2021): *On deterministic 1-limited $5' \to 3'$ sensing Watson-Crick finite-state transducers*. *RAIRO Theor. Informatics Appl.* 55, pp. 1–18, doi:10.1051/ITA/2021007.

[28] Benedek Nagy & Friedrich Otto (2020): *Linear automata with translucent letters and linear context-free trace languages*. *RAIRO Theor. Informatics Appl.* 54, p. 3, doi:10.1051/ITA/2020002.

[29] Benedek Nagy & Shaghayegh Parchami (2021): *On deterministic sensing $5' \to 3'$ Watson–Crick finite automata: a full hierarchy in 2detLIN*. *Acta Inf.* 58(3), p. 153–175, doi:10.1007/s00236-019-00362-6.

[30] Benedek Nagy & Shaghayegh Parchami (2022): *$5' \to 3'$ Watson-Crick automata languages-without sensing parameter*. *Nat. Comput.* 21(4), pp. 679–691, doi:10.1007/S11047-021-09869-9.

[31] Benedek Nagy, Shaghayegh Parchami & Hamid Mir Mohammad Sadeghi (2017): *A New Sensing $5' \to 3'$ Watson-Crick Automata Concept*. In Erzsébet Csuhaj-Varjú, Pál Dömösi & György Vaszil, editors: *Proceedings 15th International Conference on Automata and Formal Languages, AFL 2017, Debrecen, Hungary, September 4-6, 2017*, *EPTCS* 252, pp. 195–204, doi:10.4204/EPTCS.252.19.

[32] Benedek Nagy & Walaa Yasin (2025): *On some Classes of Reversible 2-head Automata*. In Nelma Moreira & Luca Prigioniero, editors: *Proceedings 15th International Workshop on Non-Classical Models of Automata and Applications (NCMA 2025), NCMA 2025, Loughborough, UK, 21-22 July 2025*, *Electronic Proceedings in Theoretical Computer Science* 422, Open Publishing Association, pp. 89–103, doi:10.4204/EPTCS.422.7.

[33] A. Nerode (1958): *Linear automaton transformations*. *Proc. Amer. Math. Soc.* 9, pp. 541–544, doi:10.1090/S0002-9939-1958-0135681-9.

[34] Shaghayegh Parchami & Benedek Nagy (2018): *Deterministic Sensing $5' \to 3'$ Watson-Crick Automata Without Sensing Parameter*. In Susan Stepney & Sergey Verlan, editors: *Unconventional Computation and Natural Computation - 17th International Conference, UCNC 2018, Fontainebleau, France, June 25-29, 2018, Proceedings*, *Lecture Notes in Computer Science* 10867, Springer, pp. 173–187, doi:10.1007/978-3-319-92435-9_13.

[35] Gheorghe Păun, Grzegorz Rozenberg & Arto Salomaa (1998): *DNA Computing - New Computing Paradigms*. Texts in Theoretical Computer Science. An EATCS Series, Springer, Heidelberg, doi:10.1007/978-3-662-03563-4.

[36] Grzegorz Rozenberg & Arto Salomaa (1997): *Handbook of Formal Languages*. Springer, doi:10.1007/978-3-642-59126-6.

[37] Kai Salomaa (2007): *Descriptional Complexity of Nondeterministic Finite Automata*. In Tero Harju, Juhani Karhumäki & Arto Lepistö, editors: *Developments in Language Theory, 11th International Conference, DLT 2007, Turku, Finland, July 3-6, 2007, Proceedings*, *Lecture Notes in Computer Science* 4588, Springer, pp. 31–35, doi:10.1007/978-3-540-73208-2_6.

[38] A.L. Semenov (1974): *Regularity of languages k-linear for various k*. *Dokl. Akad. Nauk SSSR* 215(2), pp. 278–281.

[39] José M. Sempere & Pedro García (1994): *A Characterization of Even Linear Languages and its Application to the Learning Problem*. In Rafael C. Carrasco & José Oncina, editors: *Grammatical Inference and Applications, Second International Colloquium, ICGI-94, Alicante, Spain, September 21-23, 1994, Proceedings*, *Lecture Notes in Computer Science* 862, Springer, pp. 38–44, doi:10.1007/3-540-58473-0_135.

On some Classes of Reversible 2-head Automata

Benedek Nagy
Department of Mathematics, Eastern Mediterranean University
99628 Famagusta, North Cyprus, Mersin-10, Turkey
Department of Computer Science, Institute of Mathematics and Informatics,
Eszterházy Károly Catholic University, Eger, Hungary
nbenedek.inf@gmail.com

Walaa Yasin
Department of Mathematics, Eastern Mediterranean University
99628 Famagusta, North Cyprus, Mersin-10, Turkey

Deterministic 2-head finite automata which are machines that process an input word from both ends are analyzed for their ability to perform reversible computations. This implies that the automata are backward deterministic, enabling unique forward and backward computation. We explore the computational power of such automata, discovering that, while some regular languages cannot be accepted by these machines, they are capable of accepting some characteristic linear languages, e.g., the language of palindromes. Additionally, we prove that restricted variants, i.e., both 1-limited reversible 2-head finite automata and complete reversible 2-head finite automata are less powerful and they form a proper hierarchy. In the former, in each computation step exactly one input letter is being processed, i.e., only one of the heads can read a letter. These automata are also characterized by putting their states to classes based on the head(s) used to reach and to leave the state. In the complete reversible 2-head finite automata, it is required that any input can be fully read by the automaton. The accepted families are also compared to the classes generated by left deterministic linear grammars.

1 Introduction

Formal languages can be defined using grammars or recognized through various computational devices. The latter approach is particularly valuable for determining whether a specific word belongs to a given language. These devices can operate either deterministically or non-deterministically, with the latter often being more efficient for such tasks, e.g., in terms of complexity. However, this efficiency is often offset by the reduced expressive power of non-deterministic machines.

Watson–Crick (WK) automata, introduced as part of DNA computing, combine automata theory with biological principles [35]. They operate on double-stranded tapes (representing DNA molecules), where each strand is read separately by read-only heads, maintaining the Watson–Crick complementarity relation. Various restricted classes of WK automata have been studied, with constraints on states and/or transitions. A key concept in WK automata is the $5' \to 3'$ direction, which aligns with the biochemical reading direction of DNA strands. Some models use a sensing parameter [16], which ensures that the reading heads are within a fixed distance or meet at the same position, enabling decision-making at that point. Research has shown that WK automata can characterize specific language classes, such as linear context-free languages and their subclasses (e.g., even linear languages [14]). A newer model has been proposed to eliminate the sensing parameter while maintaining the same computational power [29, 30]. It has been proven that its deterministic version accepts exactly the same class of languages (2detLIN) as the previous deterministic model with the sensing parameter [28, 31]. This class includes the class of even linear languages, which are of interest due to their learnability in formal languages [36].

N. Moreira and L. Prigioniero (Eds.): 15th International Workshop on
Non-Classical Models of Automata and Applications (NCMA 2025).
EPTCS 422, 2025, pp. 89–103, doi:10.4204/EPTCS.422.7

Further studies explore related and expanded models, such as WK counter machines [3, 4], WK automata with iterated reading [10], double and 2-head jumping automata [7, 8], translucent letter models [26, 27], 2-head pushdown automata [17, 18], 2-head automata with output [24, 25] and 2-head automata for circular words [22]. These models refine the classification of languages and computational hierarchies, providing deeper insights into automata theory and DNA computing. We should also recall various concepts of determinism that are defined for 2-head automata: state-deterministic and quasi-deterministic $5' \to 3'$ WK automata [19, 20], respectively. In computations in these models, not the next configuration, but the next state is defined uniquely based on the current state or on the current configuration, respectively.

The reversibility of standard Watson–Crick (WK) automata is an important aspect related to their computational behavior and efficiency. Since WK automata operate on double-stranded tapes with complementary base pairing, reversibility can be explored in terms of whether a computation can be uniquely reversed to reconstruct the original input [2] (with the condition that the initial state does not have any predecessor states). Now, concentrating on $5' \to 3'$ WK automata, the case is little bit different. Here, the computations follow the natural biochemical reading direction of DNA strands, and their deterministic variants ensure that each configuration leads to a unique subsequent configuration. However, for a WK automaton to be fully reversible, every transition must be invertible, meaning that for each computational step, there exists a unique predecessor configuration. This property is closely linked to bijective complementarity and sensing parameters, which influence how information is processed when the heads of the automaton meet. Investigating reversible WK automata ([2]) can provide insights into energy-efficient computations, as reversible computing is known to reduce energy dissipation, a concept relevant to both theoretical computer science and molecular computations.

In this research, the focus is on machines based on the concept of reversible deterministic 2-head finite automata. Their non-deterministic variants are introduced in [11, 15] and in [12, 30]. A non-deterministic 2-head finite automaton consists of a finite control mechanism that reads symbols from a read-only input tape using two input heads. The left (first) head scans symbols from left to right, while the right (second) head moves from right to left. During computation, the finite control non-deterministically selects one or both of the heads, read(s) the next symbol, and transitions to a new state, with the new state chosen in a non-deterministic manner. Computation concludes when the two heads finish reading the input word and meet at some point on the tape. As is standard, the input word is accepted if there exists a computation that ends in a final state when the heads meet.

Non-deterministic 2-head finite automata can recognize the class of linear context-free languages generated by such context-free grammars, where each production contains at most one non-terminal on the right-hand side. Additionally, there is a correspondence between 2-head finite automata and linear grammars, allowing for conversion between the two representations. The deterministic 2-head automata have less expressive power, the class 2detLIN is accepted by them [28]. In this paper, we address the reversibility of the computations in these models. Thus, we recall a related model from [9]. A reversible two-party Watson-Crick automaton (REV-PWK) consists of two independent finite automata that process the same input string in opposite directions while communicating via messages to ensure synchronization and reversibility.

In this paper, we consider special deterministic 2-head automata that are also backward deterministic, i.e., they are reversible. We also show that language families 2detLIN, and the classes accepted by reversible 2-head automata and their 1-limited and complete variants show a proper hierarchy, in which some of the new classes are incomparable with the class of regular languages. Some of the main results are summarized in the diagram shown in the concluding section. Note that because of the page limit some of the proofs are omitted.

2 Preliminaries and Basic Definitions

This section reviews some of the fundamental concepts in formal language and automata theory. We assume that the reader has prior knowledge of these fundamental concepts in this field. Otherwise, they may refer to sources such as [6, 34] for further details.

Let V be a finite, non-empty set, an *alphabet* of symbols, commonly referred to as letters. Sequences formed using these letters are known as *words*. A collection of such words constitutes a language over the alphabet V. The symbol λ represents the empty word.

To establish our notation, we recall that a **linear context-free grammar** is a generative grammar defined as (N,V,S,P), where N and V represent the sets of nonterminal and terminal symbols, respectively, S is the start symbol ($S \in N$), and P is the set of production rules. Each production rule (or rewriting rule) follows the form $A \to u$, where $A \in N$ and $u \in V^* + V^*NV^*$. The class of linear context-free grammars generates the family LIN, which consists of all linear context-free languages. In general, for any generative or accepting system A, we denote its generated or accepted language as $L(A)$.

2.1 2-Head Finite Automata Accepting Linear Languages and their Variants

We recall a variant of finite automata with two heads based on [11, 15] that plays essential roles in this paper. They read the word from the beginning and the end, in parallel, in opposite directions.

Definition 1 *A non-deterministic 2-head automaton is a 5-tuple (Q,q_0,V,δ,F) with the transition function*

$$\delta : Q \times (V \cup \{\lambda\}) \times (V \cup \{\lambda\}) \to 2^Q,$$

where Q is the finite set of states, $q_0 \in Q$ is the starting (or initial) state, $F \subset Q$ contains the final states, and V is the alphabet.

Computations and accepting computations are defined through configurations containing the current state and the part of the input that has not been processed yet:
When input w is provided, the computation begins with (q_0,w). A computation step $(q,aw'b) \Rightarrow (p,w')$ can occur if $p \in \delta(q,(a,b))$. The automaton processes the entire input word until the heads meet. Each letter is read by one of the heads during an accepting computation. The word w is accepted if $(q_0,w) \Rightarrow^* (q,\lambda)$ where $q \in F$.

In the graph of an automaton in transitions, each arrow is labeled with a pair of symbols (a,b), indicating that the first head reads symbol a and the second head reads symbol b, with both heads moving in one step. We allow any or both a and b to be λ. Additionally, we use the notations $\to a$ to represent (a,λ) and $\leftarrow a$ to represent (λ,a). As, actually, only those states play role in any computations that can be reached from the initial state, we may assume in the followings that all states of the used automata are reachable, i.e., there is an input such that the/a computation on this input includes a configuration containing the given state. Transitions when both heads read the empty word can easily be eliminated (somewhat similarly as λ-transitions from a traditional 1-head nondeterministic finite automaton); moreover, they are not allowed in deterministic variant (similarly as there is no λ-transition in the case of traditional deterministic finite automata).

Definition 2 *A 2-head finite automaton is deterministic if for every possible configuration, there is at most one transition step that can occur and at least one of the heads of the automaton reads an input letter in this transition (if any). In other words, a 2-head automaton is deterministic if and only if for every word $w \in V^*$ and every state $q \in Q$ there is at most one pair $w' \in V^*$ and $q' \in Q$ such that the transition $(q,w) \Rightarrow (q',w')$ is valid and $w \neq w'$.*

The deterministic version of these automata is weaker than the general, non-deterministic variant: i.e., they do not accept all linear languages. Consider the linear language $L_{1\vee 3} = \{a^n b^n\} \cup \{a^{3n} b^n\}$ ($n > 0$). It is clear that it can be accepted by a non-deterministic 2-head finite automaton trying both possibilities to check in a non-deterministic way. For a deterministic automaton, informally, it should be decided which head moves in which step. With a finite control, it is impossible to know at first how many steps of the first head should be followed by a step of the second head. For more details on the model, see e.g., [28].

Theorem 1 *A 2-head automaton (Q, q_0, V, δ, F) is deterministic if and only if the following conditions are satisfied:*

1. *For each $q \in Q$, $\delta(q, (\lambda, \lambda)) = \emptyset$.*
2. *For each $q \in Q$ and $a, b \in V \cup \{\lambda\}$, $ab \neq \lambda$ we have $|\delta(q, (a, b))| \leq 1$.*
3. *If $\delta(q, (a, \lambda)) \neq \emptyset$, where $a \in V$ and $\delta(q, (c, d)) \neq \emptyset$, then $c \neq a$ and $c \neq \lambda$.*
4. *If $\delta(q, (\lambda, a)) \neq \emptyset$, where $a \in V$ and $\delta(q, (c, d)) \neq \emptyset$, then $d \neq a$ and $d \neq \lambda$.*

Deterministic 2-head finite automata are denoted as D-2H, while the class of all languages recognized by them is denoted by 2detLIN [23, 28, 31]. Moreover, specific variants of these automata are also defined and used [13, 21, 30, 29, 31]. In *1-limited* variant, in each computation step exactly one of the heads is reading an input symbol. On the other hand, we may also define "complete" variant that is somewhat analogous to the completely defined deterministic finite automata used in the case of regular languages. We say that a D-2H automaton is *complete* if for every possible configuration (q, w) with a nonempty word there is exactly one pair $w' \in V^*$ and $q' \in Q$ such that the transition $(q, w) \Rightarrow (q', w')$ is valid. It is also clear (as it is proven in [28]) that every language in 2detLIN is accepted also by a 2-head automaton that is deterministic, 1-limited and complete.

As our main topic is reversibility, we give our first new definition. Remember that we assume that all states of the automata we consider are reachable, and therefore, the set of all possible configurations is the same as the set of configurations that appear in some computation.

Definition 3 *A 2-head finite automaton A is backward deterministic if at each possible configuration in a computation on a given input there is at most one predecessor configuration in the computations of A, i.e., in each configuration it is clear what was read in the last transition (by knowing what parts of the original input have already been processed) and from which state we arrived to the state of the current configuration.*

Theorem 2 *A 2-head finite automaton A is backward deterministic if and only if $\forall w' \in V^*$ and $\forall q' \in Q$ and $\forall a, b \in V$ there exists at most one $w \in V^*$ with $w = aw'$, $w = w'b$ or $w = aw'b$ and at most one $q \in Q$ such that $(q, w) \Rightarrow (q', w')$.*

Proof For the first direction, for every $w' \in V^*$, $q' \in Q$, and $a, b \in V$, there is at most one $w \in V^*$ such that $w = aw'$, $w = w'b$, or $w = aw'b$, and at most one $q \in Q$ such that $(q, w) \Rightarrow (q', w')$. This means that for any configuration (q', w'), there is at most one possible previous configuration (q, w) leading to it in one step if either a is read by the first head or b is read by the second head, or both at the same time. Therefore, the reverse transition relation (from (q', w') to its predecessors (q, w)) is deterministic: each configuration has at most one predecessor. This is precisely the definition of backward determinism.

For the second direction, suppose A is backward deterministic, i.e., for every configuration (q', w'), there is at most one configuration (q, w) such that $(q, w) \Rightarrow (q', w')$ knowing what was already read from the input with each head.

We now want to show that this implies the stated condition: Let us suppose, for contradiction, that for some $w' \in V^*$, $q' \in Q$, and $a,b \in V$, there exist two different pairs (q_1,w_1) and (q_2,w_2) such that $w_1,w_2 \in \{aw',w'b,aw'b\}$ and at least one of $q_1 \neq q_2$ and $w_1 \neq w_2$ holds. However, in this case, both $(q_1,w_1) \Rightarrow (q',w')$, and $(q_2,w_2) \Rightarrow (q',w')$ are valid computation steps, therefore (q',w') has at least two predecessor configurations, contradicting backward determinism.

So, the assumption that more than one such w and/or q exist leads to a contradiction. Therefore, the condition must hold. •

Now we turn to reversible 2-head finite automata, which is our main topic here. Basically, reversibility in finite automata is meant with respect to the possibility of stepping the computation back and forth. So, the machine has also to be backward deterministic.

For classical finite automata, an automaton for the reversal of the accepted language can be obtained by constructing the reversal [33], or dual automaton [32], i.e., by reversing the transitions and interchanging initial and final states. For 2-head automaton, we can obtain an automaton for the reversal of the language by simply interchanging the role of the first and second heads in each transition.

Lemma 1 *Let $A = (Q,q_0,V,\delta,F)$ be a 2-head finite automaton that recognizes a language L. Then, there exists a 2-head finite automaton $A' = (Q,q_0,V,\delta',F)$ that recognizes the reversal of L (denoted as L^R), which can be obtained by interchanging the transitions of the first and second heads in A, i.e., $\delta'(q,(a,b)) = \delta(q,(b,a))$ for all $q \in Q$ and $a,b \in V \cup \{\lambda\}$.*

Proof Since a 2-head automaton reads an input string using two separate heads, reversing the language requires adjusting how these heads process the input. By modifying the transition function such that the roles of the two heads are swapped, the automaton can effectively process the reversed string in the same manner as the original automaton processed a word of L. This ensures that A' definitely accepts L^R while preserving the computational structure of A. •

Thus, we can see that the reversal of the language is not connected to backward transitions. However, it is also interesting to see what happens, if we apply a similar construction as for finite automata to 2-head finite automata.

Definition 4 *A 2-head automaton is reversible (it is* R-2H*, for short), if it is both deterministic and backward deterministic.*

Theorem 3 *A D-2H automaton is reversible, if for any 2 transitions $\delta(q_1,(a,b)) = q_2$ and $\delta(q_1',(c,d)) = q_2$, then $a \neq c$ or $b \neq d$ with $|ad| \geq 1$ and $|bc| \geq 1$.*

Proof To prove that a deterministic 2-head automaton is reversible under the given conditions, we must establish that every transition in the automaton is uniquely reversible. Suppose that there exist two transitions: $\delta(q_1,(a,b)) = q_2$, and $\delta(q_1',(c,d)) = q_2$, we must ensure that the original states q_1 and q_1' can be uniquely determined from q_2 by knowing what was the original input. The given condition states that at least one of $a \neq c$ and $b \neq d$ holds. This ensures that no two transitions map the read input pairs (a,b) and (c,d) to the same state unless at least one symbol differs. This avoids ambiguity in the reverse of the transition. Additionally, the condition $|ad| \geq 1$ and $|bc| \geq 1$ ensures that the left head in the first transition and the right head in the second transition cannot be λ at the same time, and the same applies to the right head from the first transition and the left head from the second transition. This means at least one of the heads reads a nonempty input, a letter, in the transition, and it is decided which head moves in which step in the reversal, ensuring that reversing the transition is always possible. •

Lemma 2 *The regular language $L_{ab} = \{a^n b^m \mid n,m \geq 0\}$ cannot be accepted by* R-2H.

Proof We aim to prove that the language $L_{ab} = \{a^n b^m \mid n,m \geq 0\}$ cannot be accepted by any reversible 2-head automaton. A reversible automaton must satisfy the property that every configuration has at most one predecessor and one successor, ensuring unique invertibility of transitions.

Assume that there exists a reversible 2-head automaton $M = (Q,\{a,b\},q_0,\delta,F)$ that accepts the language $L_{ab} = \{a^n b^m \mid n,m \geq 0\}$. Let $|Q| = k$, where k is finite. The automaton M must process the input string $a^n b^m$ while maintaining reversibility. We consider the three possible cases for how M might read the input $a^n b^m$ ($n,m >> k$) in the first steps of the computation:

1. Reading a's and b's together. Suppose M uses transitions like $p = \delta(q_0,(a,b))$ to read the first a and the last b simultaneously with a state p (maybe it equals to q_0). However, as also the word $a \in L$, it must be accepted, thus there must be a transition $q' = \delta(q_0,(\lambda,a))$, since the transition $q' \in \delta(q_0,(a,\lambda))$ would contradict to the fact that M is deterministic. Also, it is clear that $q' \neq q_0$, otherwise input $a^k b^b a$ would also be accepted. On the other hand $b \in L$ must also be accepted, thus there must be a transition from the initial state to accept it. However, neither $q'' \in \delta(q_0,(b,\lambda))$, nor $q'' \in \delta(q_0,(\lambda,b))$ could be, as any of those would contradict to the determinism of M with the previously described two transitions. Thus, to process the word $a^n b^m$ with $n,m >> k$, the first computation step cannot process both an a and a b.
2. Reading only an a first. Then there is a transition $p \in \delta(q_0,(a,\lambda))$ (maybe with $p = q_0$) in M. However, M must also accept every word from b^*, thus it must also have a transition $q' \in \delta(q_0,(b,\lambda))$ (since M is deterministic, it cannot have a transition $q' \in \delta(q_0,(\lambda,b))$). If $q' = q_0$, then M would also accept a word of the form $ba^n b^m$ that leads to a contradiction. Thus, $q' \neq q_0$ must hold. In this case, however, as all other words of b^* must be accepted, there is an accepting continuation of the computation from the configuration (q',b^m) with $m > k$. As m is larger than the number of states, there is a state q'' such that from (q'',b^j) the computation also reaches the configuration (q'',b^i) with $i < j \leq m$, i.e., M has a cycle reading only b-s. Let q'' be the state for which the value of j is the maximal (among states included in the cycle). Then, there are at least two different transitions, i.e., transitions from two distinct states to q'' by reading only a b. However, in this case M cannot be reversible, causing a contradiction. (Note that q'' could be the same as q', but it cannot be q_0.)
3. Reading a b first by the transition $p \in \delta(q_0,(\lambda,b))$. The proof of this case is symmetric to the previous case, by interchanging the roles of left and right heads and the letters a and b, leading again to contradictions.

In all cases, the finite set of states of M and the requirement of reversibility create fundamental limitations. Cycles in the automaton violate reversibility by making it impossible to uniquely reconstruct the history of the computation. The deterministic nature of M further restricts its ability to handle transitions without ambiguity.

Thus, no reversible 2-head automaton can accept the regular language $L = \{a^n b^m \mid n,m \geq 0\}$. •

Based on the fact that each regular language is in 2detLIN and the previous Lemma, we can state our first hierarchy result.

Theorem 4 *The class of languages accepted by reversible 2-head automata is a proper subset of the class accepted by deterministic 2-head automata.*

$$2revLIN \subsetneq 2detLIN.$$

Proof The inclusion comes directly from the definition, as each reversible 2-head automaton is also deterministic. The properness is a consequence of Lemma 2. •

In the next sections we consider some specific variants and show that they can accept only subclasses of 2revLIN. However, first, in the next subsection, some related models are recalled.

2.2 Related Models

First, we recall a related model from [9] that we have already mentioned.

A reversible two-party Watson-Crick automaton (REV-PWK) consists of two independent finite automata that process the same input string in opposite directions, communicating via messages to ensure synchronization and reversibility. There exist reverse transition functions δ_i^- for each automaton and reverse broadcast functions μ_i^-. These functions ensure that every configuration has at most one predecessor, which can be computed using another two-party Watson-Crick system. The upper component (A_1) reads the input tape from left to right, while the lower component (A_2) reads the input tape from right to left. During a forward computation step, the upper component reads an input symbol and then moves its head to the next position, and the lower component moves its head first and then reads the input symbol. During a backward computation step, the reverse behavior occurs, the upper component either moves its head to the left or stays stationary, while the lower component either moves its head to the right or stays stationary. Two-party Watson-Crick automata are generally complex due to the need for communication and coordination between two independent components. This communication mechanism allows the automata to coordinate their actions, making the model more expressive than systems without communication. In contrast, a reversible 2-head automata, which uses a single automaton with two heads reading the input in opposite directions, are simpler because they involve a single automaton controlling both heads, as they lack the communication capability, limits its ability to handle certain languages. For instance, the language $\{a^n b^m \mid n,m \geq 0\}$ can be accepted by a REV-PWK due to its synchronization abilities but cannot be accepted by our reversible 2-head automata as we have already shown.

We also recall other models that use the generative approach.

The definition of deterministic linear grammars, as referenced in [1, 5], specifies two key properties. First, the left deterministic linear grammar does not include production rules where the right-hand side begins with a nonterminal, Additionally, for any given nonterminal T, the first terminal appearing on the right-hand side of a rule with T as the left-hand side uniquely identifies the rule, and this terminal is always followed by a nonterminal. While the right deterministic linear grammar does not include production rules where the left-hand side begins with a nonterminal, and for any given nonterminal T, the first terminal appearing on the left-hand side of a rule with T as the rigt-hand side uniquely identifies the rule, and this terminal is always followed by a nonterminal.

We recall the definition of the former special class of linear languages which were used in [1, 5], and show their relation to our reversible 2-head automata definition.

Definition 5 ***(Left Deterministic Linear Grammar)*** *A left deterministic linear grammar (LDLG)*

$$G = (N,V,S,P)$$

is a linear grammar where all rules are of the form $T \to aT'u$ or $T \to \lambda$, where for each $T,T',T'' \in N$, $u,v \in V^*$ and $a \in V$, if $T \to aT'u$, and $T \to aT''v$ are in P, then $T' = T''$ and $u = v$.

The languages generated by LDLG are called left deterministic linear language (LDLL).

Note that, in fact, the symmetric form called right deterministic linear grammars/languages (RDLG/ RDLL) are also defined in [1, 5]. Due to the symmetry of these grammars RDLG and LDLG, the reversal of any language of one of these classes is in the other class. Because of lack of space we do not go into further details about them.

Table 1: Classifying states in a reversible 1-limited 2-head automata. The first index indicates which head was used to reach the given state, the second index shows which head is allowed to read in transition(s) from the given state, where $\rightarrow$ represents the first, $\leftarrow$ represents the second head and $\emptyset$ represents the case when there is no incoming or outgoing transition from/to the state (which may happen at an initial or a final state, respectively) The trivial one-state automaton without any transitions is not considered.

to \ from q	first head	second head	NO HEAD
first head	$Q_{\rightarrow,\rightarrow}$	$Q_{\rightarrow,\leftarrow}$	$Q_{\rightarrow,\emptyset}$ (final
second head	$Q_{\leftarrow,\rightarrow}$	$Q_{\leftarrow,\leftarrow}$	$Q_{\leftarrow,\emptyset}$ state)
NO HEAD	$Q_{\emptyset,\rightarrow}$ (inital	$Q_{\emptyset,\leftarrow}$ state)	

3 Reversible Automata Consuming the Input Letterwise

In this section, we consider reversible 2-head automata with limitation on the number of heads that can move. Thus, we continue by defining a specific subset of the class R-2H automata. We have already mentioned 1-limited variants of 2-head automata, now we define formally also for the reversible case.

Definition 6 *Let A be a (deterministic/reversible) 2-head automaton. If A has the property that every transition is of the form $\delta(q,(a,\lambda))$ or every transition is of the form $\delta(q,(\lambda,b))$ $(a,b \in V)$, then A is a 1-limited (deterministic/reversible) 2-head automaton.*

The class of 1-limited reversible 2-head automata is denoted by R1-2H, while the class of languages accepted by 1-limited reversible 2-head automata is denoted by 2rev1LIN.

In fact the above definition states that for each state q of the automaton for any (not necessarily distinct) letters $a,b \in V$, $\delta(q,(a,b)) = \emptyset$ always hold.

Now let us consider these 1-limited reversible 2-head automata, where in each computation step exactly one input letter is being processed. We can classify the states of the automaton as Table 1 shows the possible properties. In fact, we can state this classification as a characterization result.

Theorem 5 *Let M be a 1-limited reversible 2-head automaton accepting a nonempty language. Then its set of states can be written as the union of at most 7 disjoint classes according to Table 1.*
The initial state q_0 can be any of the following 6 classes:

$$Q_{\rightarrow,\rightarrow}, Q_{\rightarrow,\leftarrow}, Q_{\leftarrow,\rightarrow}, Q_{\leftarrow,\leftarrow}, Q_{\emptyset,\rightarrow}, Q_{\emptyset,\leftarrow},$$

depending on if q_0 can be part of any configurations other than starting ones. Each accepting state can be in of the following classes:

$$Q_{\rightarrow,\rightarrow}, Q_{\rightarrow,\leftarrow}, Q_{\rightarrow,\emptyset}, Q_{\leftarrow,\rightarrow}, Q_{\leftarrow,\leftarrow}, Q_{\leftarrow,\emptyset}.$$

Every other state $q \in Q \setminus F \setminus \{q_0\}$ can be in one of the four classes:

$$Q_{\rightarrow,\rightarrow}, Q_{\rightarrow,\leftarrow}, Q_{\leftarrow,\rightarrow}, Q_{\leftarrow,\leftarrow}.$$

Proof We assume that all the states of Q are reachable and useful. Thus, let us start the investigation by ordinary states, i.e., if $q \in Q \setminus (F \cup \{q_0\})$. Then, as q is reachable there is a transition of the form $q \in \delta(p,(c,d))$, where exactly one of c and d is a letter of the alphabet. To make sure that M is reversible, then all transitions of the form $q \in \delta(p',(e,f))$ must also be a similar type, i.e., the same head must be

used. Moreover, if the read letter is the same, than the same state $p = p'$ must be, otherwise M would not be backward deterministic. Thus, the direction of the first arrow in the index, is already specified by this/these transition(s). Since q is useful, there is an accepting computation that contains it in a configuration. Thus, there are also transitions from q: there is a state $r \in \delta(q,(a,b))$ with some pairs such that either a or b is an input letter. However, then, each transition from state q must use the same head for reading, otherwise M would not be deterministic. Thus, the second arrow direction is also fixed. Notice that in the same automaton all the four possible states may occur.

Now, considering the initial state, if there is a transition to it, then exactly one of the above conditions apply. However, if there is no transition to the initial state, then depending on which head is allowed to read in transitions from the initial state, it is in one of the $Q_{\emptyset,\rightarrow}, Q_{\emptyset,\leftarrow}$ categories. However, no other reachable state can be in any of these categories, therefore, at most one of these categories may appear in an automaton M, but not both at the same time.

Finally, considering any of the accepting states, let us say $q \in F$: either there is at least one transition defined from q, and therefore it is in one of the four standard categories; or if no transition is defined from q, then it can be in one of the $Q_{\rightarrow,\emptyset}, Q_{\leftarrow,\emptyset}$ categories. Notice that, as M may have many accepting states, in an automaton both of these special categories may occur.

Thus, we can see, that M as a 1-limited reversible 2-head automaton, can have at most 7 disjoint classes of states. •

Of course, the possible transitions are also constrained based on the categories of the states. From a state where the second arrow direction is $\ell \in \{\rightarrow,\leftarrow\}$ all transitions are going to states where the first arrow is also ℓ.

To make complete the characterization, we also state the following:

Theorem 6 *A 1-limited 2-head automaton M is reversible if and only if it has the following properties.*

- *The states of M can be classified according to the classes of Theorem 5.*
- *M is deterministic: for each state q and each letter a, there is at most one transition, i.e.,* $|\delta(q,(a,\lambda))| \leq 1$ *for states where the first head is allowed to read, and* $|\delta(q,(\lambda,a))| \leq 1$ *for states where the second head is allowed to read.*
- *M is backward deterministic, i.e., for each state q and each letter a, there is at most one transition to arrive to q, i.e., there is at most one state p such that* $q \in \delta(p,(a,\lambda))$ *for state q that can be reached by transition(s) in which the first head is allowed to read, and there is at most one state p such that* $q \in \delta(p,(\lambda,a))$ *for a state q that can be reached by transition(s) in which the second head is allowed to read.*

Proof This theorem is the consequence of the definition and the previous characterization result. •

Now, we intend to show that although the class of 1-limited variant of D-2H is able to accept all languages in 2detLIN, for reversible automata the 1-limited variant cannot be used as a normal form, as they are weaker than the variants without such restriction.

Lemma 3 *The language* $L_{wcb^n} = \{wcb^n \mid w \in \{a,b\}^*,\ |w|_b = n\}$ *is not accepted by any R1-2H automaton.*

We can now conclude the following hierarchy result:

Theorem 7 *The languages accepted by R1-2H is a proper subclass of the languages accepted by R-2H:*

$$2rev1LIN \subsetneq 2revLIN.$$

Proof The inclusion comes directly from the definitions. The properness can be proven, on the one hand, considering the R-2H automaton with two states as follows: at the initial state q_0 let there be two loop transitions: by (b,b) and by (a,λ). Further, let a transition by (c,λ) from q_0 to q_f which is the final state. Easy to see that this automaton is reversible. This R-2H automaton accepts the language

$$L_{wcb^n} = \{wcb^n \mid w \in \{a,b\}^*,\ |w|_b = n\}.$$

On the other hand, as it was shown in Lemma 3 this language is not in 2rev1LIN. •

4 Completely defined Reversible 2-Head Automata

In this section we investigate another subset of R-2H based on the already mentioned "complete" restriction.

Definition 7 *A deterministic/reversible 2-head finite automaton is complete if for every possible configuration, there is exactly one transition step that can occur, if the input is not yet fully processed. In other words, a deterministic/reversible 2-head automaton is complete if and only if for every nonempty word $w \in V^* \setminus \{\lambda\}$ and every state $q \in Q$ there is exactly one pair $w' \in V^*$ and $q' \in Q$ such that the transition $(q,w) \Rightarrow (q',w')$ is valid.*

The class of reversible complete 2-head automata is denoted by RC-2H, while the class of languages accepted by them is denoted as 2CrevLIN.

Now, on the one hand, in [28] it is proven that each language of 2detLIN can be accepted by a complete deterministic 2-head automaton. (Actually, for each language of 2detLIN there is also a complete 1-limited deterministic 2-head automaton that accepts it.) On the other hand, our aim, in this section, is to show that the property "completeness" is a real restriction for reversible 2-head automata.

Proposition 1 *There is no complete reversible 2-head finite automaton that has any transition of the form: $\delta(q,(a,b)) = q'$ for some $a,b \in V$. That is, no complete reversible 2-head automaton can have any transition where both heads move simultaneously.*

Proof Let $A = (Q,q_0,V,\delta,F)$ be a complete reversible 2-head finite automaton (and as we always assume with only states that are reachable). We shall prove that if $\delta(q,(a,b)) = q'$ for some $a,b \in V$, then A cannot be complete. Assume for contradiction that A has the transition: $\delta(q,(a,b)) = q'$.

This means that in state q, the first head can read an a and the second head reads a b at the same time while the automaton switches its state to q'.

Let us consider an input uv such that state q is reached from q_0 by reading u with the first and v with the second head, respectively. Such input exists by our assumption. Then, we state that A cannot completely read at least one of the original inputs uav and ubv. The computation on these inputs goes as $(q_0,uav) \Rightarrow^* (q,a)$ and $(q_0,ubv) \Rightarrow^* (q,b)$, respectively. Considering the first word, the current configuration is (q,a). To be able to read a in the next step of the computation A must have at least one of the transitions $\delta(q,(a,\lambda)) \neq \emptyset$ or $(q,(\lambda,a)) \neq \emptyset$. Actually, as A is deterministic, we can be sure that not both such transitions exist in A. Moreover, as we already assumed that $\delta(q,a,b) \neq \emptyset$, thus $\delta(q,(a,\lambda)) \neq \emptyset$ is impossible as it would also contradict to the deterministic behavior of A. Thus, we can conclude that to allow to process the input uav completely, A must have $(q,(\lambda,a)) \neq \emptyset$.
Now, let us consider the configuration (q,b). To make the input ubv completely read by A, we need to able to read this b at state q. For that we would need at least one of $\delta(q,(b,\lambda)) \neq \emptyset$ or $\delta(q,(\lambda,b)) \neq \emptyset$.

As A is deterministic, in fact, exactly one of those. However, in case $\delta(q,(b,\lambda)) \neq \emptyset$, we have a contradiction: this with $\delta(q,(\lambda,a)) \neq \emptyset$ cannot be in a deterministic 2-head automaton.
Further, the second transition $(q,(\lambda,b)) \neq \emptyset$ cannot be in A as it would contradict to the deterministic behavior of A since it has the transition $\delta(q,(a,b)) \neq \emptyset$.

In a similar manner, it can also be seen that it is also impossible in a complete reversible automaton if $a = b$, i.e., the transition in which both heads read is of the form $\delta(q,(a,a)) \neq \emptyset$.

Therefore, we can conclude that a transition where both heads move requires an input of length greater than 2. But completeness demands that the automaton must run on any nonempty input, including length 1. Thus, such a transition cannot be used on all inputs, violating completeness. •

Thus, one may see that completeness automatically involve the 1-limited restriction.

We have the following structural result about these automata.

Theorem 8 *Let A be a complete 1-limited reversible 2-head automaton. Then the graph of A is strongly connected, i.e., there is a computation for each pair of states p and q such that if the current configuration has state p, then there is a (remaining) input such that by processing it, A reaches a configuration with state q.*

Proof For each state the number of defined transitions is exactly the same as the cardinality of the alphabet, let us say n. Since these automata are reversible, there is no state to which more than n transition could be defined. However, in this case, there are exactly n "incoming" transitions to each state. Thus, for any subset Q' of states, the number of transitions coming from "outside" (i.e., from a state in $Q \setminus Q'$ to a state in Q') must be the same as the number of transitions going outside from Q' (i.e., transitions from a state in Q' to a state in $Q \setminus Q'$). From this it follows that from any state p that is reachable, there is also a computation to the initial state q_0. •

Lemma 4 *The language $L_{ba} = \{b^n a^n, b^{n+1} a^n \mid n \geq 0\}$ cannot be accepted by any complete reversible 2-head automaton.*

Proof Let us assume that there is a complete reversible 2-head automaton A that accepts L_{ba}. Then, in its initial state q_0 either head must start the process. If the first head starts to read, then A must have both transitions with (a,λ) and (b,λ) to ensure completeness. However, there is no word in L_{ba} that starts with a letter a, thus by the former transition a state should be reached from which no accepting computation can be continued. However, this violates the strongly connected property of A. Thus, A must start with the second head, by having both transitions with (λ,a) and (λ,b). This second should reach an accepting state, as $b \in L_{ba}$. However, as no other words having suffix b are in the language, from this accepting state each continuation of the computation must not be accepting. As the automaton is complete, it should able to read any unread input, but again, not to accept any continuations would lead to a contradiction to the strongly connectedness of A. •

Theorem 9 *The class of all languages that can be accepted by reversible complete 2-head automata (RC-2H) is a proper subset of 2revLIN. Furthermore, it is a proper subclass of the class accepted by 1-limited reversible 2-head automata:*

$$2CrevLIN \subsetneq 2rev1LIN.$$

Proof It is clear that RC-2H is a subset of R-2H, and thus 2CrevLIN is a subset of 2revLIN from the definitions. Moreover, RC-2H is also a subset of R1-2H by Proposition 1. The strict inclusion comes from Lemma 4 as the language $L_{ba} = \{b^n a^n, b^{n+1} a^n \mid n \geq 0\}$ can clearly be accepted by a 1-restricted,

but not complete reversible 2-head automaton with 2 states, let us say q_0 and q, having a transition from q_0 to q by reading a b with the first head, and having a transition from q to q_0 by reading an a with the second head. Both states are also final states. •

Further, about the closure properties of the language class accepted by RC-2H automata we have the following.

Theorem 10 *The class 2CrevLIN of languages accepted by complete (1-limited) 2-head reversible automata is closed under complementation operation.*

Proof As an RC-2H automaton $A = (Q, q_0, V, \delta, F)$ can fully read every input in a deterministic manner, the automaton $\bar{A} = (Q, q_0, V, \delta, Q \setminus F)$ accepts exactly the complement of $L(A)$. Since only the accepting states are changed $\bar{A}$ also belongs to the class RC-2H. •

We close this section with another closure property that we have for each our new classes.

Theorem 11 *Each of the classes 2detLIN, 2revLIN, 2rev1LIN and 2CrevLIN of languages, i.e., the classes accepted by deterministic 2-head automata, reversible 2-head automata, by 1-limited reversible 2-head automata and by complete (1-limited) 2-head reversible automata, respectively, is closed under reversal operation.*

Proof Observe that the construction used in Lemma 1 does not modify any of the following properties: determinism, reversibility, 1-limitedness and completeness. Therefore, if the original automaton has any of these properties, the automaton that accepts the reversal of the language has also the same restrictions. •

5 Discussion and Summary

In this section, we start with a relation among the language classes accepted by our deterministic/reversible 2-head automata and those that are generated by left deterministic linear grammars (Definition 5).

Theorem 12 *Every left deterministic linear language is accepted by deterministic 2-head automata, moreover, the inclusion LDLL $\subsetneq$ 2detLIN is proper.*

We present various example languages in Fig. 1 (some of the proofs about them are left for the reader).

Proposition 2 *The regular language $L_{ab} = \{a^n b^m\}$ is clearly in LDLL.*

Further, in this section, we summarize the results concerning the relations between the considered classes on Fig. 1. We have proven the proper hierarchy

$$2CrevLIN \subsetneq 2rev1LIN \subsetneq 2revLIN \subsetneq 2detLIN.$$

Moreover, as we have seen, there are regular languages that are not in 2revLIN. However, still 2rev1LIN contains, e.g., the language of palindromes.

There are some open problems as well. There are some regions of the diagram where we do not put any example language (see ? marks in the figure). It is a future task to find examples to fit there, or to prove if some of these regions are empty. Based on our examples, we conjecture that 2*CrevLIN* contains only regular languages. Decidability problems, computational and descriptional complexity issues regarding the new classes are also open and can be studied in the future.

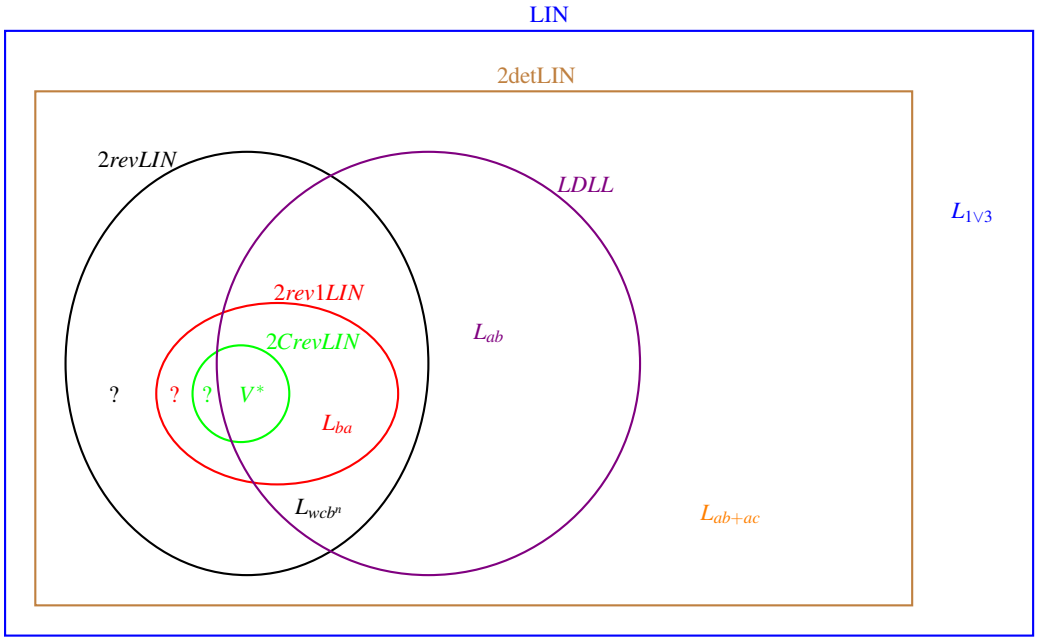

Figure 1: Diagram of sublinear languages with the example language V^* and other examples from the paper.

Acknowledgments

The authors are very grateful to the reviewers for their comments.

References

[1] Jorge Calera-Rubio & Jose Oncina (2004): *Identifying Left-Right Deterministic Linear Languages*. In: *Grammatical Inference: Algorithms and Applications (ICGI 2004)*, *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)* 3264, pp. 283–284, doi:10.1007/978-3-540-30195-0_29.

[2] Kingshuk Chatterjee & Kumar Sankar Ray (2017): *Reversible Watson—Crick automata*. *Acta Inf.* 54(5), p. 487–499, doi:10.1007/s00236-016-0267-0.

[3] Ömer Egecioglu, László Hegedüs & Benedek Nagy (2011): *Hierarchies of Stateless Multicounter* $5' \to 3'$ *Watson-Crick Automata Languages*. *Fundam. Informaticae* 110(1-4), pp. 111–123, doi:10.3233/FI-2011-531.

[4] László Hegedüs, Benedek Nagy & Ömer Egecioglu (2012): *Stateless multicounter* $5' \to 3'$ *Watson-Crick automata: the deterministic case*. *Nat. Comput.* 11(3), pp. 361–368, doi:10.1007/S11047-011-9290-9.

[5] Colin de la Higuera & Jose Oncina (2002): *Inferring Deterministic Linear Languages*. In: *Computational Learning Theory (COLT 2002)*, *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)* 2375, pp. 185–200, doi:10.1007/3-540-45435-7_13.

[6] John E. Hopcroft & Jeffrey D. Ullman (1979): *Introduction to Automata Theory, Languages and Computation*. Addison Wesley. Available at https://api.semanticscholar.org/CorpusID:31901407.

[7] Radim Kocman, Zbynek Krivka & Alexander Meduna (2016): *On double-jumping finite automata*. In Henning Bordihn, Rudolf Freund, Benedek Nagy & György Vaszil, editors: *Eighth Workshop on Non-Classical*

Models of Automata and Applications, NCMA 2016, Debrecen, Hungary, August 29-30, 2016. Proceedings, *books@ocg.at* 321, Österreichische Computer Gesellschaft, pp. 195–210.

[8] Radim Kocman, Zbynek Krivka, Alexander Meduna & Benedek Nagy (2022): *A jumping $5' \to 3'$ Watson-Crick finite automata model*. *Acta Informatica* 59(5), pp. 557–584, doi:10.1007/S00236-021-00413-X.

[9] Martin Kutrib & Andreas Malcher (2023): *Reversible Two-Party Computations*. In Zsolt Gazdag, Szabolcs Iván & Gergely Kovásznai, editors: *Proceedings of the 16th International Conference on Automata and Formal Languages, AFL 2023, Eger, Hungary, September 5-7, 2023*, *EPTCS* 386, pp. 142–154, doi:10.4204/EPTCS.386.12.

[10] Peter Leupold & Benedek Nagy (2010): *$5' \to 3'$ Watson-Crick AutomataWith Several Runs*. *Fundam. Informaticae* 104(1-2), pp. 71–91, doi:10.3233/FI-2010-336.

[11] Roussanka Loukanova (2007): *Linear Context Free Languages*. In Cliff B. Jones, Zhiming Liu & Jim Woodcock, editors: *Theoretical Aspects of Computing - ICTAC 2007, 4th International Colloquium, Macau, China, September 26-28, 2007, Proceedings*, *Lecture Notes in Computer Science* 4711, Springer, pp. 351–365, doi:10.1007/978-3-540-75292-9_24.

[12] Benedek Nagy (2008): *On $5' \to 3'$ Sensing Watson-Crick Finite Automata*. In Max H. Garzon & Hao Yan, editors: *DNA Computing, 13th International Meeting on DNA Computing, DNA13, Memphis, TN, USA, June 4-8, 2007, Revised Selected Papers*, *Lecture Notes in Computer Science* 4848, Springer, pp. 256–262, doi:10.1007/978-3-540-77962-9_27.

[13] Benedek Nagy (2009): *On a hierarchy of $5' \to 3'$ sensing WK finite automata languages*. In: *Mathematical Theory and Computational Practice, CiE 2009, Abstract Booklet, Heidelberg, Germany*, pp. 266–275.

[14] Benedek Nagy (2010): *$5' \to 3'$ Sensing Watson-Crick Finite Automata*. In: *Sequence and Genome Analysis II — Methods and Applications*, iConcept Press, pp. 39–56.

[15] Benedek Nagy (2012): *A class of 2-head finite automata for linear languages*. *Triangle* 8, pp. 89–99.

[16] Benedek Nagy (2013): *On a hierarchy of $5' \to 3'$ sensing Watson–Crick finite automata languages*. *Journal of Logic and Computation* 23(4), pp. 855–872, doi:10.1093/logcom/exr049. arXiv:https://academic.oup.com/logcom/article-pdf/23/4/855/2775832/exr049.pdf.

[17] Benedek Nagy (2015): *A family of two-head pushdown automata*. In Rudolf Freund, Markus Holzer, Nelma Moreira & Rogério Reis, editors: *Seventh Workshop on Non-Classical Models of Automata and Applications - NCMA 2015, Porto, Portugal, August 31 - September 1, 2015. Proceedings*, *books@ocg.at* 318, Österreichische Computer Gesellschaft, pp. 177–191.

[18] Benedek Nagy (2020): *$5' \to 3'$ Watson-Crick pushdown automata*. *Inf. Sci.* 537, pp. 452–466, doi:10.1016/J.INS.2020.06.031.

[19] Benedek Nagy (2021): *State-deterministic $5' \to 3'$ Watson-Crick automata*. *Nat. Comput.* 20(4), pp. 725–737, doi:10.1007/S11047-021-09865-Z.

[20] Benedek Nagy (2022): *Quasi-deterministic $5' \to 3'$ Watson-Crick Automata*. In Henning Bordihn, Géza Horváth & György Vaszil, editors: *Proceedings 12th International Workshop on Non-Classical Models of Automata and Applications, NCMA 2022, Debrecen, Hungary, August 26-27, 2022*, *EPTCS* 367, pp. 160–176, doi:10.4204/EPTCS.367.11.

[21] Benedek Nagy (2023): *On language classes accepted by stateless $5' \to 3'$ Watson-Crick finite automata*. *Annales Mathematicae et Informaticae* 58, pp. 110–120, doi:10.33039/ami.2023.08.004.

[22] Benedek Nagy (2024): *$5' \to 3'$ Watson-Crick Automata accepting Necklaces*. In Florin Manea & Giovanni Pighizzini, editors: *Proceedings 14th International Workshop on Non-Classical Models of Automata and Applications (NCMA 2024), NCMA 2024, Göttingen, Germany, 12-13 August 2024*, *EPTCS* 407, pp. 168–185, doi:10.4204/EPTCS.407.12.

[23] Benedek Nagy (2025): *A Myhill-Nerode type characterization of 2detLIN languages*. In Nelma Moreira & Luca Prigioniero, editors: *Proceedings 15th International Workshop on Non-Classical Models of Automata and Applications, NCMA 2025, Loughborough, UK, 21-22 July 2025*, *Electronic Proceedings in Theoretical Computer Science* 422, Open Publishing Association, pp. 73–88, doi:10.4204/EPTCS.422.6.

[24] Benedek Nagy & Zita Kovács (2019): *On simple $5' \to 3'$ sensing Watson-Crick finite-state transducers*. In Rudolf Freund, Markus Holzer & José M. Sempere, editors: *Eleventh Workshop on Non-Classical Models of Automata and Applications, NCMA 2019, Valencia, Spain, July 2-3, 2019*, Österreichische Computer Gesellschaft, pp. 155–170.

[25] Benedek Nagy & Zita Kovács (2021): *On deterministic 1-limited $5' \to 3'$ sensing Watson-Crick finite-state transducers*. *RAIRO Theor. Informatics Appl.* 55, pp. 1–18, doi:10.1051/ITA/2021007.

[26] Benedek Nagy & Friedrich Otto (2019): *Two-Head Finite-State Acceptors with Translucent Letters*. In Barbara Catania, Rastislav Královič, Jerzy Nawrocki & Giovanni Pighizzini, editors: *SOFSEM 2019: Theory and Practice of Computer Science*, Lecture Notes in Computer Science, Springer International Publishing, Cham, pp. 406–418, doi:10.1007/978-3-030-10801-4_32.

[27] Benedek Nagy & Friedrich Otto (2020): *Linear automata with translucent letters and linear context-free trace languages*. *RAIRO Theor. Informatics Appl.* 54, p. 3, doi:10.1051/ITA/2020002.

[28] Benedek Nagy & Shaghayegh Parchami (2021): *On deterministic sensing $5' \to 3'$ Watson–Crick finite automata: a full hierarchy in 2detLIN*. *Acta Inf.* 58(3), p. 153–175, doi:10.1007/s00236-019-00362-6.

[29] Benedek Nagy & Shaghayegh Parchami (2022): *$5' \to 3'$ Watson-Crick automata languages-without sensing parameter*. *Nat. Comput.* 21(4), pp. 679–691, doi:10.1007/S11047-021-09869-9.

[30] Benedek Nagy, Shaghayegh Parchami & Hamid Mir Mohammad Sadeghi (2017): *A New Sensing $5' \to 3'$ Watson-Crick Automata Concept*. In Erzsébet Csuhaj-Varjú, Pál Dömösi & György Vaszil, editors: *Proceedings 15th International Conference on Automata and Formal Languages, AFL 2017, Debrecen, Hungary, September 4-6, 2017*, *EPTCS* 252, pp. 195–204, doi:10.4204/EPTCS.252.19.

[31] Shaghayegh Parchami & Benedek Nagy (2018): *Deterministic Sensing $5' \to 3'$ Watson-Crick Automata Without Sensing Parameter*. In Susan Stepney & Sergey Verlan, editors: *Unconventional Computation and Natural Computation - 17th International Conference, UCNC 2018, Fontainebleau, France, June 25-29, 2018, Proceedings*, *Lecture Notes in Computer Science* 10867, Springer, pp. 173–187, doi:10.1007/978-3-319-92435-9_13.

[32] Jean-Eric Pin (1992): *On reversible automata*. In Imre Simon, editor: *LATIN '92*, Springer, Berlin, Heidelberg, pp. 401–416, doi:10.1007/BFb0023844.

[33] Michael O. Rabin & Dana Scott (1959): *Finite automata and their decision problems*. *IBM journal of research and development* 3(2), pp. 114–125, doi:10.1147/rd.32.0114.

[34] Grzegorz Rozenberg & Arto Salomaa (1997): *Handbook of Formal Languages*. Springer, doi:10.1007/978-3-642-59126-6.

[35] Grzegorz Rozenberg & Arto Salomaa (1999): *DNA Computing: New Ideas and Paradigms*. In Jirí Wiedermann, Peter van Emde Boas & Mogens Nielsen, editors: *Automata, Languages and Programming*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 106–118, doi:10.1007/3-540-48523-6_9.

[36] José M. Sempere & Pedro García (1994): *A Characterization of Even Linear Languages and its Application to the Learning Problem*. In Rafael C. Carrasco & José Oncina, editors: *Grammatical Inference and Applications, Second International Colloquium, ICGI-94, Alicante, Spain, September 21-23, 1994, Proceedings*, *Lecture Notes in Computer Science* 862, Springer, pp. 38–44, doi:10.1007/3-540-58473-0_135.