Algebraic Characterizations of Classes of Regular Languages in DynFO

Corentin Barloy
Ruhr University Bochum, Germany

Felix Tschirbs
Ruhr University Bochum, Germany

Nils Vortmeier
Ruhr University Bochum, Germany

Thomas Zeume
Ruhr University Bochum, Germany

Abstract

This paper explores the fine-grained structure of classes of regular languages maintainable in fragments of first-order logic within the dynamic descriptive complexity framework of Patnaik and Immerman. A result by Hesse states that the class of regular languages is maintainable by first-order formulas even if only unary auxiliary relations can be used. Another result by Gelade, Marquardt, and Schwentick states that the class of regular languages coincides with the class of languages maintainable by quantifier-free formulas with binary auxiliary relations.

We refine Hesse's result and show that with unary auxiliary data $\exists^*\forall^*$-formulas can maintain all regular languages. We then obtain precise algebraic characterizations of the classes of languages maintainable with quantifier-free formulas and positive $\exists^*$-formulas in the presence of unary auxiliary relations.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic; Theory of computation → Algebraic language theory; Theory of computation → Regular languages

Keywords and phrases Dynamic descriptive complexity, formal languages, monoid theory

Digital Object Identifier 10.4230/LIPIcs.STACS.2026.9

Related Version *Full Version*: `https://arxiv.org/abs/2601.18429`

Funding All authors are supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), grant 532727578.

Acknowledgements We thank the reviewers for their careful reading and insightful comments, which significantly improved the quality of this article.

1 Introduction

In seminal work by Schützenberger as well as McNaugthon and Papert, the class of regular languages has been characterized algebraically as the class of languages with finite syntactic monoids [16] and logically as the class of languages definable by monadic second-order sentences [1]. In subsequent work, a multitude of algebraic and logical characterisations have been obtained for subclasses of regular languages, including a characterization of the class of star-free regular languages (languages describable by regular expressions without Kleene star but with negation) as the class of languages with finite aperiodic monoids and definable in first-order logic [17, 9]; and the characterization of piecewise testable languages (languages describable by the set of subwords appearing in the word) as languages with $\mathcal{J}$-trivial syntactic monoids and definable by first-order formulas without quantifier alternation [18].

43rd International Symposium on Theoretical Aspects of Computer Science (STACS 2026).
Editors: Meena Mahajan, Florin Manea, Annabelle McIver, and Nguyễn Kim Thắng
Article No. 9; pp. 9:1–9:19

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Here we are interested in algebraic properties of formal languages defined in the dynamic descriptive complexity framework of Patnaik and Immerman [10]. In this framework, strings are subject to changes, and the goal is to maintain membership within a language via logical formulas, possibly using auxiliary information stored in auxiliary relations (which need to be updated by logical formulas as well). An important class of languages in this setting are the languages maintainable via first-order formulas, called DynFO. It has been shown early on that DynFO contains all regular and even all context-free languages [10, 4].

The goal of this paper is to initiate a search for connections between algebra and dynamic descriptive complexity classes. This is in a similar spirit as the search for connections between algebra and logics initiated by Schützenberger, McNaugthon and Papert for the static realm, as sketched above. Our guiding question is:

> *Are there restrictions of* DynFO *that lead to natural subclasses of the regular languages with nice algebraic characterizations?*

For answering this question, it is natural to first identify restrictions of DynFO that can still maintain all regular languages. This has been studied in prior work. A result by Hesse states that the class of regular languages is maintainable by first-order formulas even if only very restricted auxiliary information, namely only unary relations, can be used [6, Theorem 2.8][1]. A seminal result by Gelade, Marquardt, and Schwentick states that, if binary auxiliary relations can be used, then the class of regular languages coincides with the class DynProp of languages maintainable by quantifier-free formulas [4, Theorem 3.2].

We refine the result by Hesse. Denote by $\mathsf{UDyn}\Sigma_2$ the class of languages maintainable by $\exists^*\forall^*$-formulas using unary auxiliary relations.

▶ **Theorem 1.** *All regular language are in* $\mathsf{UDyn}\Sigma_2$.

This result can be obtained by close inspection of Hesse's proof, which maintains properties of the execution graph of finite state automata. Here, we provide an algebraic reformulation of the proof relying on Green's relations (see Section 3).

Thus, when looking for restrictions of DynFO that correspond to natural subclasses of the regular languages with nice algebraic characterizations, one can explore (a) fragments of (binary) DynProp, or (b) fragments of $\mathsf{UDyn}\Sigma_2$.

We obtain results along both directions. For binary DynProp, it is natural to further restrict the auxiliary relations. We precisely characterize the languages in DynProp with unary auxiliary relations by algebraic properties of their syntactic monoids (see Section 4).

▶ **Theorem 2.** *A regular language is in* UDynProp *if any only if its syntactic monoid is a group.*

For $\mathsf{UDyn}\Sigma_2$, it is natural to further restrict the syntactic structure of formulas used for updates. We precisely characterize the languages maintainable by positive $\exists^*$-formulas, denoted $\mathsf{UDyn}\Sigma_1^+$, by the algebraic properties of their ordered syntactic monoids (see Section 5).

▶ **Theorem 3.** *A regular language is in* $\mathsf{UDyn}\Sigma_1^+$ *if and only if its ordered syntactic monoid is in* $\mathbf{J}^+ * \mathbf{G}$.

Here, $\mathbf{J}^+ * \mathbf{G}$ denotes the wreath product of the classes $\mathbf{J}^+$ and $\mathbf{G}$.

[1] A variant of this result has already been reported in [10], but additional built-in arithmetic has been used as auxiliary information.

In the proofs of both Theorem 2 and Theorem 3, the "if" direction requires to provide update formulas for all regular languages whose (ordered) syntactic monoids satisfy some property and is standard. The "only-if" direction requires to prove lower bounds against DynProp (resp. UDynΣ_1^+) for all non-group languages (resp. all languages whose ordered syntactic monoid is not in $\mathbf{J}^+ * \mathbf{G}$). To this end we prove a lower bound for a single language using variants of the Substructure lemma [4] (see also [20]) and lift this lower bounds to all languages without the property using algebraic insights.

We leave open the question to find a precise algebraic characterization of languages in UDynΣ_1, i.e., languages maintainable by (not necessarily positive) $\exists^*$-formulas and unary auxiliary relations. We discuss challenges towards answering this question in Section 6.

2 Preliminaries

In this section, also to fix notation, we recall basics of dynamic descriptive complexity and algebraic formal language theory.

2.1 Dynamic descriptive complexity

In this paper, we are interested in the dynamic membership problem for fixed regular languages. Let $L \subseteq \Sigma^*$ be a language over some alphabet Σ. The input structure for the dynamic membership problem $\textsc{Member}(L)$ of L is an encoding of a word $w = w_1 \cdots w_n$ as a relational structure over the domain $\{1, \ldots, n\}$ of the positions of the word. This structure has (1) for every $\sigma \in \Sigma$ a unary relation W_σ for storing the positions of w that contain σ and (2) a binary relation $\leq$ representing the linear order of the positions of w. Following Patnaik and Immerman as well as Gelade et al. [10, 4], each w_i is either in Σ or ϵ, i.e., each position $i \in [n]$ can occur in at most one relation W_σ. For the sake of simplicity, we denote both input words and their encoding by the same symbol w.

In the dynamic membership problem, the input structure can be changed by operations of the form $\textsc{set}_\sigma(y)$, for $\sigma \in \Sigma \cup \{\epsilon\}$. A concrete change $\textsc{set}_\sigma(i)$ instantiates y and sets the symbol at position i to σ. The dynamic membership problem $\textsc{Member}(L)$ asks whether the current input structure encodes a word in L, i.e. whether $w_1 \cdots w_n \in L$.

We will explore the resources needed for maintaining the dynamic membership problem within the dynamic descriptive complexity framework of Patnaik and Immerman [10].

A *dynamic program* Π stores the input structure as well as a set $\mathcal{A}$ of auxiliary relations over some (relational) schema τ_{aux} and over the same domain as the input word. For every auxiliary relation symbol $R \in \tau_{\text{aux}}$ and every change operation $\textsc{set}_\sigma$, the program Π has an *update formula* $\varphi_\sigma^R(\bar{x}; y)$, which can access input and auxiliary relations. Whenever the current input word w is changed by $\textsc{set}_\sigma(i)$ to the new input word w', the new auxiliary relation $R^{\mathcal{A}'}$ in the updated set $\mathcal{A}'$ consists of all tuples $\bar{a}$ such that $\varphi_\sigma^{\mathcal{A}}(\bar{a}; i)$ is satisfied in the structure $(w', \mathcal{A})$.

A dynamic program Π maintains the dynamic membership problem $\textsc{Member}(L)$ if it has a distinguished auxiliary bit q (i.e., a 0-ary auxiliary relation), which always indicates whether the current input word is in L. More precisely, following Patnaik and Immerman, we assume that the initial input word w_0 is empty and the initial auxiliary structure $\mathcal{A}_0$ is initialized by a first-order formula depending on w_0. The program Π *maintains* $\textsc{Member}(L)$, if after changing w_0 to w' by any sequence of changes and subsequently applying the corresponding update formulas starting from $(w_0, \mathcal{A}_0)$, the obtained bit q is true if and only if $w' \in L$.

The dynamic problem $\textsc{Member}(L)$ is in the class DynFO if it can be maintained by a dynamic program with FO update formulas and an FO initialization formula. It is in k-ary DynFO if all auxiliary relations are at most k-ary. For a class $\mathcal{C}$ of formulas, it is in Dyn$\mathcal{C}$ if all update formulas are in $\mathcal{C}$ and the initialization formula is in FO.

Throughout this paper we assume that formulas are in prenex normal form. Among the classes $\mathcal{C}$ we study are the class Prop of quantifier-free formulas; the class Σ_1 of $\exists^*$-formulas (i.e., formulas in prenex form with only existential quantifiers); and the class Σ_2 of $\exists^*\forall^*$-formulas (i.e., formulas in prenex form starting with a block of existential quantifiers followed by a block of universal quantifiers). We also consider the variants Prop^+, Σ_1^+ and Σ_2^+ where negations may only be used directly in front of the linear order symbol $\leq$. A focus will be on programs that can only use unary auxiliary relations, in which case we denote classes by $\mathsf{UDyn}\mathcal{C}$ for classes $\mathcal{C}$ of update formulas.

▶ **Example 4.** We show that the regular language $\Sigma^*\mathtt{aa}\Sigma^*$ over $\Sigma = \{a, b\}$ is in $\mathsf{UDyn}\Sigma_1$, so, can be maintained using only unary auxiliary relations and update formulas in prenex form that only use existential quantifiers.

Our goal is to maintain a unary relation N that stores all positions j such that the next letter to the right of j that does not contain ϵ is an a. This relation can directly be maintained using both Σ_1 and Π_1 update formulas. As Π_1 update formulas are not allowed, we store a slightly differently defined relation N' and an additional bit c. If the flag c is true, the relation N' is equal to N. If c is false, then N' is the complement of N', that is, contains all position j such that either all positions to the right of j are ϵ or the first positions that is not ϵ is a b. Using this relation, we can obtain N as $N(j) = (\neg c \wedge \neg N'(j)) \vee (c \wedge N'(j))$.

We now describe the update formulas for maintaining N' and c. We additionally use unary auxiliary relations W_σ^o to store the old input word before the change. These are trivial to maintain.

- In case of a change of the form $\textsc{set}_b(i)$, we set $c = 1$ and the relation N' after the update is equal to N. A position j is in N' if $j \in N$ before the change and either $i \leq j$ or there is a position between i and j that is an a. The corresponding update formula for N' is $\varphi_b^{N'}(j; i) = \exists k\big[N(j) \wedge \big(i \leq j \vee (j < i \wedge j < k \wedge k < i \wedge W_a(k))\big)\big]$, where $x < y$ is an abbreviation for $x \leq y \wedge \neg(x = y)$.
- In case of a change of the form $\textsc{set}_a(i)$, we set $c = 0$ and the relation N' after the update is the complement of N. A position j is in N' if $j \notin N$ before the change and either $i \leq j$ or there is a position between i and j that is a b. The corresponding update formula for N' is $\varphi_a^{N'}(j; i) = \exists k\big[\neg N(j) \wedge \big(i \leq j \vee (j < i \wedge j < k \wedge k < i \wedge W_b(k))\big)\big]$.
- In case of a change $\textsc{set}_\epsilon(i)$, the update depends on the old symbol at position i. If the symbol was an a, the relation N' is equal to N after the update and we set $c = 1$. If the removed a was the first letter to the right of j, then we have to check the first letter to the right of i. Otherwise, the change is not relevant for position j. So, if $j < i$, the next letter from j is an a if and only if $N(j)$ and $N(i)$ were both true before the change, or there is an a between j and i and $N(j)$ was already true.
  If the change replaces b with ϵ, the relation N' is the complement of N after the update and we set $c = 0$. If $j < i$, the next letter from j is not an a if and only if $N(j)$ and $N(i)$ were both false before the change, or there is a b between j and i and $N(j)$ was false. The update formula is

$$\begin{aligned}
&\varphi_\epsilon^{N'}(j; i) = \exists k\Big[\\
&W_a^o(i) \wedge \Big((i \leq j \wedge N(j)) \vee \big(j < i \wedge N(j) \wedge (N(i) \vee (j < k \wedge k < i \wedge W_a(k)))\big)\Big) \vee \\
&W_b^o(i) \wedge \Big((i \leq j \wedge \neg N(j)) \vee \big(j < i \wedge \neg N(j) \wedge (\neg N(i) \vee (j < k \wedge k < i \wedge W_b(k)))\big)\Big)\Big].
\end{aligned}$$

Similarly, we can maintain a unary auxiliary relation P that contains all positions j such that the next letter to the left of j that does not contain ϵ is an a.

We now describe the maintenance of the bit q that is set to 1 whenever w is in $L = \Sigma^* aa \Sigma^*$.

- In case of a change $\text{SET}_a(i)$, the new word w is in L if and only if q or one of $N(i)$ and $P(i)$ was already true before the change.
- In case of a change $\text{SET}_b(i)$, we have $w \in L$ if and only if there exists a position $k > i$ with an a such that $N(k)$ holds, or there exists a position $k < i$ with an a such that $P(k)$ holds.
- In case of a change $\text{SET}_\epsilon(i)$ it holds $w \in L$ if and only if there exists a position $k > i$ with an a such that $N(k)$ holds, or there exists a position $k < i$ with an a such that $P(k)$ holds, or both $N(i)$ and $P(i)$ hold.

These conditions can easily be described by existential update formulas. ⌟

During our study, we will need the following closure properties. For a language L over alphabet Σ and a letter $\sigma \in \Sigma$, we define $L\sigma^{-1} = \{w \mid w\sigma \in L\}$ and $\sigma^{-1}L = \{w \mid \sigma w \in L\}$.

▶ **Lemma 5.** *Let L be a language over alphabet Σ and $\mathsf{DynC} \in \{\mathsf{UDynProp}, \mathsf{UDyn}\Sigma_1^+\}$. If $\text{MEMBER}(L)$ can be maintained in DynC, then so can*

(a) $\text{MEMBER}(h^{-1}(L))$ *for any mapping $h : \Gamma \to \Sigma^*$ and any alphabet Γ,*

(b) $\text{MEMBER}(L\sigma^{-1})$ *and* $\text{MEMBER}(\sigma^{-1}L)$*, for any $\sigma \in \Sigma$.*

2.2 Monoids

For DynFO and many of its natural restrictions that possess favourable closure properties, we follow Skovberg Frandsen, Miltersen, and Skyum [19] in studying dynamic problems for monoids rather than languages. A *monoid* $(M, \cdot)$ is a set M equipped with a binary operation $\cdot$ that is associative and has an identity element, i.e., the operation satisfies $(x \cdot y) \cdot z = x \cdot (y \cdot z)$ for all elements $x, y, z \in M$ and there is an element $1 \in M$ such that $1 \cdot x = x \cdot 1 = x$ for all $x \in M$. A monoid is a *group* if every element x has an inverse, i.e. an element y such that $x \cdot y = y \cdot x = 1$. An element e is said to be *idempotent* if $e \cdot e = e$. For an integer $n > 0$ and $x \in M$, the n-fold product of x is denoted by x^n. It is well known that every element $x \in M$ has a unique power that is idempotent [12, Prop. II.6.31] called *idempotent power* of x and denoted by x^ω.

A *morphism* $\varphi : M \to N$ between two monoids is a mapping such that $\varphi(1) = 1$ and $\varphi(x \cdot y) = \varphi(x) \cdot \varphi(y)$ for every $x, y \in M$. A *submonoid* N of a monoid M is a subset of M that is a monoid, i.e., it contains 1 and is closed under $\cdot$, meaning $x \cdot y \in N$ for all $x, y \in N$. A monoid N is a *quotient* of M if there exists a surjective morphism from M to N. We say that N *divides* M if N is the quotient of a submonoid of M. A language L is *recognized* by a monoid M if there is a morphism $\varphi : \Sigma^* \to M$ and $P \subseteq M$ such that $L = \varphi^{-1}(P)$. The regular languages are precisely those recognized by a finite monoid (see for instance [12, Theorem IV.3.21]).

For a monoid M, we consider dynamic programs over the alphabet M and define the problem of evaluating a sequence of elements in M. We say that a dynamic program Π maintains the dynamic membership problem $\text{MEMBER}(M)$ if there are $|M|$ dedicated auxiliary bits that respectively store for every $x \in M$ whether the maintained word evaluates to x. Similarly the dynamic problem StrictPrefix (resp. StrictSuffix) is maintained via one dedicated unary relation for each $x \in M$ that stores the value $1 \leq i \leq n$ if the prefix $w_1 \cdot \ldots \cdot w_{i-1}$ (resp. suffix $w_{i+1} \cdot \ldots \cdot w_n$) evaluates to x. In this setting, the symbol ϵ and the neutral element 1 of M can be used interchangeably. For instance, the initial word w_0 can either be the empty word or 1^n.

An important monoid is the monoid of functions $f : Q \to Q$ for a set Q. The operation is the composition and the neutral element is the identity function. Given a deterministic finite state automaton with transition function $\delta : \Sigma^* \to (Q \to Q)$, its *transition monoid* is

the image $\delta(\Sigma^*)$. The *syntactic monoid* of a regular language is the transition monoid of its minimal automaton. It is the smallest monoid that recognizes the language and reflects many properties of its regular language. Indeed, the syntactic monoid possesses more information than its regular language, as exposed by the following claim. A class of formulas $\mathcal{C}$ is closed under $\vee$ (resp. $\wedge$, resp. $\neg$) if for every $\varphi, \psi \in \mathcal{C}$, we also have $\varphi \vee \psi$ (resp. $\varphi \wedge \psi$, resp. $\neg\varphi$) in $\mathcal{C}$. All classes under inspection in this paper possesses this property.

▶ **Fact 6.** *Let $\mathcal{C}$ be a class of formulas closed under $\vee$. Let L be a regular language with syntactic monoid M and k a number, then*

$$\textsc{Member}(M) \in k\text{-ary } \mathsf{Dyn}\mathcal{C} \Rightarrow \textsc{Member}(L) \in k\text{-ary } \mathsf{Dyn}\mathcal{C}.$$

We will see in Section 4 and Section 5 that when the class of formulas under consideration is well-behaved, the converse of this claim is also true.

We conclude this section by stating that maintainability of $\textsc{Member}(M)$ for a monoid implies maintainability of $\textsc{Member}(N)$ for every N that divides M.

▶ **Fact 7.** *Let $\mathcal{C}$ be a class of formulas closed under $\vee$ and k any number. Let M be a monoid such that $\textsc{Member}(M) \in k\text{-ary } \mathsf{Dyn}\mathcal{C}$. Then for every N that divides M we have $\textsc{Member}(N) \in k\text{-ary } \mathsf{Dyn}\mathcal{C}$.*

3 The regular languages of UDynΣ_2

In this section we prove that every regular language can be maintained by Σ_2-formulas and only unary relations. That is, we prove the following.

▶ **Theorem 8** (Restatement of Theorem 1)**.**
For any regular language L: $\textsc{Member}(L)$ *is in* $\mathsf{UDyn}\Sigma_2$.

The proof is algebraic by nature, hence we show that $\textsc{Member}(M)$ for a syntactic monoid M can be maintained and then apply Fact 6. The approach towards proving Theorem 8 is as in Hesse [6]: We want to exhibit maintainable unary relations that allow to compute the evaluation of every infix of a word from M^*. This is formalized in the following lemma.

▶ **Lemma 9.** *Let M be a finite monoid. There is a dynamic program Π that maintains unary auxiliary relations over a schema τ_M such that*

- **i)** *Π only uses Σ_2 update formulas, and*
- **ii)** *for every $x \in M$ there is a Σ_2 formula $\psi_x(j,k)$ over the schema of Π that is satisfied if and only if the infix $w_{j+1} \cdots w_{k-1}$ of the input word w evaluates to x.*

The definition of the auxiliary relations and the proof of the lemma relies on the local theory of monoids, as developed by Green [5]. We first show that Theorem 8 follows.

Proof of Theorem 8. We prove that $\textsc{Member}(M)$ is in $\mathsf{UDyn}\Sigma_2$ for the syntactic monoid of a regular language. As $\mathsf{UDyn}\Sigma_2$ is closed under disjunction, this implies the result due to Fact 6. Towards this end, let Π be the dynamic program and ψ_x the formulas expressing the evaluation of infixes of the input word, as given by Lemma 9. Note that the formulas ψ_x give the evaluation of the input word ahead of a change. To maintain $\textsc{Member}(M)$, we need the evaluation after a change. Suppose a change $\textsc{set}_\sigma(i)$ occurs. The formula

$$\psi_x^\sigma(j,k;i) = \Big[(j < i < k) \wedge \Big(\bigvee_{\substack{y,z \in M \\ y\sigma z = x}} \psi_y(j,i) \wedge \psi_z(i,k) \Big)\Big] \vee \Big[(i \leq j \vee k \leq i) \wedge \psi_x(j,k)\Big]$$

then expresses that the infix from position $j+1$ to position $k-1$ of the changed input word evaluates to x. As the formulas ψ_x, ψ_y, ψ_z are not in the scope of a negation and no further quantifiers are used, the formula is also in Σ_2.

Now, the formula

$$\varphi(i) = \exists j, k\Big[\min(j) \wedge \max(k) \wedge \Big(\bigvee_{\substack{v,y,z \in M \\ yvz = x}} W_y(j) \wedge W_z(k) \wedge \psi_v^\sigma(j,k;i)\Big)\Big],$$

with $\min(j) = \forall i(i \leq j \to i = j)$ and $\max(k) = \forall i(i \geq k \to i = k)$ expresses that w evaluates to x after the change on position i, for any fixed $x \in M$. This formula still has only one alternation of quantifiers and belongs to Σ_2. Together with the update formulas of Π, we have a dynamic $\mathsf{Dyn}\Sigma_2$ program for $\textsc{Member}(M)$. ◀

In the remainder of this section, we recall Green's relations and use them to define the unary relations contained in the schema τ_M of the program Π from Lemma 9. The proof that these relations satisfy conditions (i) and (ii) from Lemma 9 is delegated to the full version of this paper.

Green's relations. The local theory of finite monoids studies the structure of monoids by considering properties of sets of elements that behave the same way. It is based on the so-called Green's relations, which are a generalization of the notion of divisibility for any monoid. Over the integers, the divisibility relation is a partial order, whereas Green's relations for arbitrary monoids are preorders, that is, reflexive and transitive relations. Fix a monoid M. Its *Green's preorders* are defined by, for $x, y \in M$:

- $x \leq_{\mathcal{R}} y$ whenever there exists $\alpha \in M$ such that $x = y\alpha$,
- $x \leq_{\mathcal{L}} y$ whenever there exists $\alpha \in M$ such that $x = \alpha y$,
- $x \leq_{\mathcal{J}} y$ whenever there exists $\alpha, \beta \in M$ such that $x = \alpha y \beta$,
- $x \leq_{\mathcal{H}} y$ whenever $x \leq_{\mathcal{R}} y$ and $x \leq_{\mathcal{L}} y$.

The *Green's classes* $\mathcal{R}$, $\mathcal{L}$, $\mathcal{J}$ and $\mathcal{H}$ of M are the equivalence classes of the preorders. For instance, two elements $x, y \in M$ are equivalent for $\mathcal{R}$, denoted by $x\mathcal{R}y$, if and only if $x \leq_{\mathcal{R}} y$ and $y \leq_{\mathcal{R}} x$. It follows from the definitions that $\mathcal{H} \subseteq \mathcal{R} \subseteq \mathcal{J}$ and $\mathcal{H} \subseteq \mathcal{L} \subseteq \mathcal{J}$. Note that for any green relation $\mathcal{K}$ among the four, $\leq_{\mathcal{K}}$ induces a partial order on the set of $\mathcal{K}$-classes. The order $\leq_{\mathcal{J}}$ is of particular interest: assume there is an element x in some $\mathcal{J}$-class J. Then, for any $y, z \in M$, yxz belongs to a $\mathcal{J}$-class I with $I \leq_{\mathcal{J}} J$. Therefore, if a factor of a word $w_1 \cdots w_n \in M^*$ evaluates to some element of a $\mathcal{J}$-class J, the whole word evaluates to an element from a class J' with $J' \leq_{\mathcal{J}} J$. There is an analogous statement for $\mathcal{R}$-classes and $\mathcal{L}$-classes that we will use repeatedly throughout the paper: if an element x is a prefix of another element y, that is there exists α such that $x\alpha = y$, then we have $y \leq_{\mathcal{R}} x$. Likewise, if x is a suffix of y, so there exists α such that $\alpha x = y$, then $y \leq_{\mathcal{L}} x$.

▶ **Example 10.** The syntactic monoid of $(\mathtt{aa})^*$ is $\mathbb{Z}_2$, the cyclic group with two elements 1 and a such that $a \cdot a = 1$. Both elements are $\mathcal{H}$-equivalent (and therefore also $\mathcal{R}$-, $\mathcal{L}$-, and $\mathcal{J}$-equivalent). In fact, in groups, all elements are always $\mathcal{H}$-equivalent.

▶ **Example 11.** The syntactic monoid of $(\mathtt{a}+\mathtt{b})^*\mathtt{a}(\mathtt{a}+\mathtt{b})^*$ is usually denoted by U_1. It has two elements 1 and a such that $a \cdot x = x \cdot a = a$ for every element x. The two elements are in distinct classes for all of Green's relations, and $1 \geq_{\mathcal{J}} a$.

▶ **Example 12.** The syntactic monoid of $(\mathtt{a}+\mathtt{b})^*\mathtt{a}$ is usually denoted by U_2. It has three elements 1, a and b such that a and b are idempotent and $a \cdot b = b$ and $b \cdot a = a$. The elements a and b are $\mathcal{R}$-equivalent (and therefore also $\mathcal{J}$-equivalent), but not $\mathcal{L}$-equivalent. The neutral element 1 is alone in its $\mathcal{J}$-class which is $\mathcal{J}$-smaller than the one of a and b.

Unary auxiliary relations for expressing evaluations of infixes. The proof of Lemma 9 relies on the fact that finite monoids have only finitely many $\mathcal{J}$-classes. Therefore, when we evaluate a word from left to right, there is only a finite number of changes of $\mathcal{J}$-classes. We make that intuition explicit. In the following, for a word $w = w_1 \cdots w_n \in M^*$, we write $w[j,k]$ to denote the subword $w_j \cdots w_k$ of w between indices j and k. For an element x, we denote by $\mathcal{J}(x)$ its $\mathcal{J}$-class. When it is clear from the context, we identify a word in M^* by its evaluation. The following lemma is about the evaluation of a word $w \in M^*$ from left to right, that is evaluating w_1, w_1w_2, $w_1w_2w_3$, ..., $w_1 \cdots w_n$ in that order. It states that, except for a constant number of times (depending on M and not on w), the $\mathcal{J}$-class of the partial evaluations does not change.

▶ **Lemma 13.** *Let w be a word from M^* of size n and $j \leq n$ a position. Then there exist integers $j = \ell_0 < \ell_1 < \cdots < \ell_m = n+1$ with $m \leq |M|$ such that:*

i) *for $0 \leq s < m$ and $\ell_s \leq i < \ell_{s+1}$, $w[j,\ell_s] \mathcal{J} w[j,i]$,*

ii) *for $0 \leq s < m-1$, $w[j,\ell_{s+1}-1] \not\mathcal{J} w[j,\ell_{s+1}]$.*

There is a symmetric statement from right to left. That is, there exists integers $0 = \ell_m < \cdots < \ell_1 < \ell_0 = j$ with $m \leq |M|$ such that:

i) *for $0 \leq s < m$ and $\ell_{s+1} < i \leq \ell_s$, $w[\ell_s,j] \mathcal{J} w[i,j]$,*

ii) *for $0 \leq s < m-1$, $w[\ell_{s+1}-1,j] \not\mathcal{J} w[\ell_{s+1},j]$.*

The relations promised in Lemma 9 contain the information, for every position i, of the evaluation of the prefixes obtained in the decomposition of Lemma 13 applied to i. In the following, we slightly abuse notation and write $x \geq_{\mathcal{J}} J$ to mean that the $\mathcal{J}$-class of x is at least J with respect to the order of $\mathcal{J}$-classes.

Let w be a word from M^*, i a position in the word, $y \in M$ and J a $\mathcal{J}$-class of M. Let k be the greatest position such that $y \cdot w[j+1,k] \geq_{\mathcal{J}} J$. Then we define $R_{\geq J,y}(j)$ as the evaluation of $w[j+1,k]$ (notice that this is *not* $y \cdot w[j+1,k]$). Similarly, let k be the least position such that $w[k,j-1] \cdot y \geq_{\mathcal{J}} J$. Then we write $L_{\geq J,y}(j)$ for the evaluation of $w[k,j-1]$. When $y = 1$, we simply write $R_{\geq J}$ and $L_{\geq J}$. We also define $\bar{R}_{\geq J,y}$ and $\bar{L}_{\geq J,y}$ for the variants that include j in the intervals. Note that it is not clear how to compute one from the other without access to successor predicates.

▶ **Definition 14.** *Let M be a monoid. The schema τ_M is defined as the following set of relations, where $x, y \in M$ and J is a $\mathcal{J}$-class of M:*

- *$R_{\geq J,y,x}$ that contains j if and only if $R_{\geq J,y}(j) = x$,*
- *$L_{\geq J,y,x}$ that contains j if and only if $L_{\geq J,y}(j) = x$,*
- *their variants $\bar{R}_{\geq J,y,x}$ and $\bar{L}_{\geq J,y,x}$.*

These unary relations satisfy conditions (i) and (ii) of Lemma 9, see the full version. The idea to compute the evaluation of an infix $w]j,k[$ with these relations is to use them to compute the decomposition of $w]j,n]$ given by Lemma 13. This can be done by existentially quantifying the positions $l_0, \ldots, l_m$ and checking the properties of the decomposition using uniquely universal quantifiers. The first l_s after k allows to deduce the $\mathcal{R}$-class of $w]j,k[$: indeed, $w]j,l_s[$ is $\mathcal{J}$-equivalent to $w]j,k[$ by the definition of the decomposition, which is strengthened into $\mathcal{R}$-equivalence because one is a prefix of the other. Using the second decomposition of Lemma 13, from rigth to left, we also deduce the $\mathcal{L}$-class of $w]j,k[$. This identify the $\mathcal{H}$-class of the infix, then more technical work is needed to compute precisely the evaluation.

4 The regular languages of UDynProp

If binary auxiliary relations are allowed, the quantifier-free fragment DynProp of DynFO is powerful enough to maintain all regular languages [4]. In this section, we provide a characterization of the regular languages that can be maintained using quantifier-free update formulas if only unary auxiliary relations are permitted. These are precisely the regular languages that are recognized by a group. Intuitively, a computation in a group can be reversed at any moment using the presence of inverses. That is, if a computation on xy ends in some memory state, it is possible to retrieve the memory state after the computation of x. We denote by **G** this class of languages.

▶ **Theorem 15** (Restatement of Theorem 2). *For any regular language L:* $\textsc{Member}(L) \in$ UDynProp *if and only if the syntactic monoid of L is a group.*

We first show that every regular group language can be maintained in UDynProp, using a simple adaptation of the dynamic program that maintains all regular languages using binary auxiliary relations. Then we prove that other regular languages cannot be maintained. For this, we adapt a lower bound result from [21] to show that UDynProp cannot maintain the regular language "there is at least one a" and then generalize this result to all regular languages whose syntactic monoid is not a group.

4.1 Maintaining regular group languages with quantifier-free formulas

We show that unary auxiliary relations and quantifier-free update formulas are sufficient to maintain the evaluation problems of groups. The upper bound from Theorem 15 follows.

▶ **Lemma 16.** *If G is a group, then the problems* $\textsc{Member}(G)$, $\textsc{StrictPrefix}(G)$ *and* $\textsc{StrictSuffix}(G)$ *are in* UDynProp.

Proof sketch. Gelade et al. [4] explain how to maintain membership, strict prefixes and suffixes for all regular languages with binary auxiliary relations. Their approach can directly be used to maintain these problems for monoids. For every monoid element g, the corresponding update formulas use a bit Q_g to indicate whether the whole input evaluates to g and three further auxiliary relations:

- a unary relation P_g that contains all positions i such that the prefix ending at position $i-1$ evaluates to g,
- a unary relation S_g that contains all positions i such that the suffix starting at position $i+1$ evaluates to g, and
- a binary relation I_g that contains all pairs (i,j) of positions such that the infix starting at position $i+1$ and ending at position $j-1$ evaluates to g.

Assume that the element at position p is set to the monoid element g_1. Let i, j be positions with $i < p-1$ and $j > p+1$. After the change, the infix from position $i+1$ to position $j-1$ evaluates to the element $g_0 \cdot g_1 \cdot g_2$, where g_0 is the result of evaluating the infix from position $i+1$ to position $p-1$ and g_2 is the result of evaluating the infix from position $p+1$ to position $j-1$. Both elements g_0 and g_2 can be read from the auxiliary relations of the form I_g without using quantifiers, given the position p of the change and the positions i, j for which the auxiliary relations are updated.

If the monoid is a group, we can replace atoms $I_g(i,j)$ in the update formulas as follows. Let g_w be the result of evaluating the whole input and let g_p, g_s be the evaluations of the prefix up to position $i-1$ and the suffix starting from position $j+1$, respectively. Let g_i and

g_j be the elements at position i and j. All of these elements can be obtained from the input and auxiliary relations without using quantifiers, given i and j. From $g_w = g_p \cdot g_i \cdot g \cdot g_j \cdot g_s$, for the evaluation g of the infix from position $i+1$ to position $j-1$, we obtain g as $g = g_i^{-1} \cdot g_p^{-1} \cdot g_w \cdot g_s^{-1} \cdot g_j^{-1}$, as every group element has an inverse.

So, we replace any atom $I_g(i,j)$ by the formula $\bigvee_{g=g_i^{-1}\cdot g_p^{-1}\cdot g_w\cdot g_s^{-1}\cdot g_j^{-1}} Q_w \wedge W_{g_i}(i) \wedge W_{g_j}(j) \wedge P_{g_p}(i) \wedge S_{g_s}(j)$ and obtain update formulas that only use at most unary auxiliary relations. The result follows. ◀

4.2 Lower bound for regular non-group languages

We want to show that Lemma 16 is optimal and any language whose syntactic monoid is not a group cannot be maintained in UDynProp. The proof is in three steps. First, we show that a specific regular language cannot be maintained in UDynProp. We generalize this and show that UDynProp cannot maintain the membership problem for any monoid that is not a group. At last, we infer that one cannot maintain any regular language whose syntactic monoid is not a group by exploiting known closure properties of this class of languages.

4.2.1 A regular language that is not in UDynProp

Schwentick and Zeume [21] have shown that the graph reachability problem for so-called 1-layer graphs – edges may only exist from the start node s to some set A of nodes and from this set A to the target node t – is not in UDynProp. Very similarly, one can show that one cannot maintain in UDynProp whether a set is empty, where changes add elements to the set or remove elements from it. We adapt the proof to show that the regular language $(\mathtt{a}+\mathtt{b})^*\mathtt{a}(\mathtt{a}+\mathtt{b})^*$, that is, the language of all words over the alphabet $\{\mathtt{a},\mathtt{b}\}$ that contain an $\mathtt{a}$, is also not in UDynProp. The subtle difference between these problems is that the positions of a word are linearly ordered, but the elements of a set are not. However, despite this additional structure, basically the same proof goes through. We provide some more detail, also because we want to adapt the proof techniques in Section 5.

As proof technique we use the Substructure Lemma [4], which basically says that if two substructures including their auxiliary relations are isomorphic and subject to changes that respect that isomorphism, then also the auxiliary relations that result from applying quantifier-free update formulas are isomorphic.

We make this more formal. For any natural number n, we write $[n]$ for the set $\{1,\dots,n\}$. If w is a word of some length n, I is a subset of $[n]$ and $\mathcal{A}$ is a set of auxiliary relations, we write $(w,\mathcal{A})_{|I}$ to denote the restriction of $(w,\mathcal{A})$ to the positions I. Let w, w' be words of length $n \leq m$ respectively and let π be a mapping from $[n]$ to $[m]$. We call π *order-preserving* if $i < i'$ implies $\pi(i) < \pi(i')$. Two sequences $\alpha = \delta_1 \cdots \delta_\ell$ and $\alpha' = \delta'_1 \cdots \delta'_\ell$ of changes of w and w', respectively, are *π-respecting* if for every pair δ_i, δ'_i of changes it holds that if $\delta_i = \textsc{set}_\sigma(j)$ for some symbol σ and some position j, then $\delta'_i = \textsc{set}_\sigma(\pi(j))$.

If α is a sequence of changes and Π is a dynamic program, we denote by $\Pi_\alpha(w,\mathcal{A})$ the pair of word and auxiliary relations that is obtained by applying the changes α and the corresponding update formulas from Π to the word w and the auxiliary relations $\mathcal{A}$.

▶ **Lemma 17** (Substructure Lemma [4, Lemma 1]). *Let Π be a dynamic* UDynProp *program and let $w \in \Sigma^n$ and $w' \in \Sigma^m$ be two words over the alphabet Σ. Further, let $\mathcal{A}$ and $\mathcal{A}'$ be sets of at most unary auxiliary relations over the domain of w and w' and the schema of Π. Assume there are $I \subseteq [n]$ and $I' \subseteq [m]$ such that $(w,\mathcal{A})_{|I}$ and $(w',\mathcal{A}')_{|I'}$ are isomorphic via some order-preserving isomorphism π. Then π is also an order-preserving isomorphism from $\Pi_\alpha(w,\mathcal{A})_{|I}$ to $\Pi_{\alpha'}(w',\mathcal{A}')_{|I'}$, for any π-respecting sequences α and α' of changes.*

To obtain a lower bound via the Substructure Lemma, we need to find isomorphic substructures such that two sequences of isomorphism-respecting changes lead to words that differ with respect to membership in a regular language. To find such structures, we use Higman's Lemma.

▶ **Lemma 18** (see [8, Chapter 6]). *Let $(w_i, \mathcal{A}_i)_{i\geq 1}$ be a sequence of words $w_i \in \Sigma^i$ and sets $\mathcal{A}_i$ of relations over a unary schema and over the domain of w_i. There exists two integers $n < m$, a subset $I \subseteq [m]$ and a mapping $\pi : [n] \to I$ such π is a order-preserving isomorphism from w_n to $w_{m|I}$.*

We now show the desired result.

▶ **Lemma 19** (c.f. [21, Proposition 4.8]). *The language $(\mathtt{a}+\mathtt{b})^*\mathtt{a}(\mathtt{a}+\mathtt{b})^*$ is not in* UDynProp.

Proof. Assume towards a contradiction that there is a dynamic UDynProp program Π that maintains $(\mathtt{a}+\mathtt{b})^*\mathtt{a}(\mathtt{a}+\mathtt{b})^*$. We consider the sequence $(w_i, \mathcal{A}_i)_{i\geq 1}$, where w_i is the word $\mathtt{a}^i$ and $\mathcal{A}_i$ are auxiliary relations that Π obtains when starting from an empty input word of size i and setting all positions to $\mathtt{a}$. By Lemma 18, there are $n < m$, $I \subseteq [m]$ and π such that π is an order-preserving isomorphism from $(w_n, \mathcal{A}_n)$ to $(w_m, \mathcal{A}_m)_{|I}$.

The sequences $\alpha = \textsc{set}_b(1)\cdots\textsc{set}_b(n)$ and $\alpha' = \textsc{set}_b(\pi(1))\cdots\textsc{set}_b(\pi(n))$ are π respecting, so according to Lemma 17, the structures $\Pi_\alpha(w_n, \mathcal{A}_n)$ and $\Pi_{\alpha'}(w_m, \mathcal{A}_m)_{|I}$ are also isomorphic, which means that Π gives the same answer regarding whether the words are in the language $(\mathtt{a}+\mathtt{b})^*\mathtt{a}(\mathtt{a}+\mathtt{b})^*$. But the first word only consists of $\mathtt{b}$ after the changes are applied, so does not belong to the language, while the second word still contains at least one $\mathtt{a}$, contradicting the assumption that Π maintains the language. ◀

4.2.2 UDynProp cannot maintain any regular non-group language

We now lift the unmaintainability result of Lemma 19 to all regular languages whose syntactic monoid is not a group. First, we observe that UDynProp cannot maintain the membership problem of any monoid that is not a group.

▶ **Lemma 20.** *If M is a monoid that is not a group, then* $\textsc{Member}(M)$ *is not in* UDynProp.

Proof. The syntactic monoid U_1 of the language $(\mathtt{a}+\mathtt{b})^*\mathtt{a}(\mathtt{a}+\mathtt{b})^*$ has only the elements 1 and a, where 1 is the neutral element and a acts as a zero: $1\cdot x = x\cdot 1 = x$ and $a\cdot x = x\cdot a = a$ for every $x \in \{1, a\}$. As a has no inverse, U_1 is not a group. It follows from Fact 6 and Lemma 19 that $\textsc{Member}(U_1)$ is not in UDynProp.

Now assume that for some arbitrary non-group monoid M the problem $\textsc{Member}(M)$ is in UDynProp. We first claim that U_1 is a submonoid of M. To see that, observe that M must have an element x whose idempotent power is not the identity. Otherwise, if $x^\omega = 1$ for every element $x \in M$, then let $n > 0$ be chosen such that $x^n = 1$ is the idempotent. Then $x \cdot x^{n-1} = x^{n-1} \cdot x = 1$ holds, so x has the inverse x^{n-1}, so M is a group, contradicting the assumption. Therefore, M has an idempotent e that is different from 1 and $\{1, e\}$ is a submonoid of M that is isomorphic to U_1.

As U_1 is a submonoid of M and in particular divides M, assuming $\textsc{Member}(M) \in$ UDynProp leads to a contradiction, as from Fact 7 the false statement $\textsc{Member}(U_1) \in$ UDynProp would follow. ◀

Now we have to translate Lemma 20 from monoids to languages, that is, we need a converse to Fact 6. So, we have to prove that if UDynProp can maintain some regular language, it can also maintain the membership problem of its syntactic monoid.

We employ the theory of (pseudo)varieties, which goes back to Eilenberg [3].

Let $\mathcal{L}$ be a class of regular languages over the alphabet Σ. It is *closed under Boolean operations* if $L_1 \cup L_2$, $L_1 \cap L_2$ and L_1^c are in $\mathcal{L}$ for any $L_1, L_2 \in \mathcal{L}$. It is *closed under quotients* if $\sigma^{-1}L = \{w \mid \sigma w \in L\}$ and $L\sigma^{-1} = \{w \mid w\sigma \in L\}$ are in $\mathcal{L}$ for any $L \in \mathcal{L}$ and $\sigma \in \Sigma$. It is *closed under inverse morphisms* if $\mu^{-1}(L)$ is in $\mathcal{L}$ for any $L \in \mathcal{L}$, where Γ is any alphabet and $\mu : \Gamma \to \Sigma^*$ is a function that is extended to Γ^* letter-wise.

A class $\mathcal{V}$ of regular languages is a *(pseudo)variety* if it is closed under Boolean operations, quotients and inverse morphisms. We often omit the "pseudo" and simply call such classes "varieties". For this work, the following result is used.

▶ **Lemma 21** ([3], see [12, Theorem XIII.4.10]). *Let $\mathcal{V}$ be a variety of regular languages and let M be the syntactic monoid of some language $L \in \mathcal{V}$. Then any language recognized by M is also in $\mathcal{V}$.*

This implies that the converse of Fact 6 holds for UDynProp.

▶ **Fact 22.** *Let $\mathcal{C}$ be a class of formulas such that the regular languages L with* $\textsc{Member}(L) \in \mathsf{UDyn}\mathcal{C}$ *form a variety. Let L be a regular language and M its syntactic monoid:*

$$\textsc{Member}(L) \in \mathsf{UDyn}\mathcal{C} \Rightarrow \textsc{Member}(M) \in \mathsf{UDyn}\mathcal{C}.$$

Proof. Assume that $\textsc{Member}(L)$ is in $\mathsf{UDyn}\mathcal{C}$ and that M is its syntactic monoid. For each $x \in M$, the monoid M recognizes the language of all sequences of monoid elements that evaluate to x, so by Lemma 21, this language is also in $\mathsf{UDyn}\mathcal{C}$. A dynamic $\mathsf{UDyn}\mathcal{C}$ program for $\textsc{Member}(M)$ maintains these languages for each $x \in M$. ◀

The "only if" direction of Theorem 15 follows: observe that the regular languages of UDynProp form a variety, as UDynProp is trivially closed under Boolean operations, but also closed under quotients and inverse morphisms due to Lemma 5. Therefore, by Fact 22, if UDynProp can maintain a regular language, it can also maintain the membership problem of its corresponding syntactic monoid. But, by Lemma 20, UDynProp cannot maintain this problem for any monoid that is not a group.

5 The regular languages of $\mathsf{UDyn}\Sigma_1^+$

In this section, we provide a characterization of the regular languages that can be maintained using positive Σ_1-formulas if only unary auxiliary relations are permitted. Here, positive Σ_1-formulas are $\exists^*$-formulas that may not use negations in the quantifier-free part, with the only exception that negations can be used directly in front of the built-in linear order $\leq$.

An inspection of the proof of Theorem 15 shows that positive quantifier-free formulas and therefore in particular Σ_1^+-formulas can maintain all regular languages recognized by groups. It is also known that Σ_1^+-formulas can express all regular languages recognized by the class $\mathbf{J}^+$ of ordered monoids (see Section 5.1 for a definition). It turns out that languages maintainable in $\mathsf{UDyn}\Sigma_1^+$ are exactly those recognizable by ordered monoids from the wreath product $\mathbf{J}^+ * \mathbf{G}$ of $\mathbf{J}^+$ and $\mathbf{G}$. Intuitively, this class contains monoids that first do a "group computation" followed by a "$\mathbf{J}^+$-computation" (see Section 5.1 for a formalization).

▶ **Theorem 23** (Restatement of Theorem 3). *For any regular language L:* $\textsc{Member}(L) \in \mathsf{UDyn}\Sigma_1^+$ *if and only if the ordered syntactic monoid of L is in $\mathbf{J}^+ * \mathbf{G}$.*

We will show, in Section 5.2, that every regular language recognized with $\mathbf{J}^+ * \mathbf{G}$ is in $\mathsf{UDyn}\Sigma_1^+$. Then, in Section 5.3, we prove that other regular languages cannot be maintained.

To this end, we first show that the language $\mathtt{b}^*$ cannot be maintained, using an adaptation of the substructure lemma to $\mathsf{UDyn}\Sigma_1^+$. Finally, relying on a characterization of $\mathbf{J}^+ * \mathbf{G}$, we show that if another language could be maintained, then so could the language $\mathtt{b}^*$. We therefore look at monoids supplemented with additional ordering information.

5.1 Ordered monoids and wreath products

For the characterization obtained in Section 4, we exploited that the class of languages maintainable in $\mathsf{UDynProp}$ is a variety and can therefore be studied by looking at the syntactic monoids of its languages. Unfortunately, the class $\mathsf{UDyn}\Sigma_1^+$ of languages is not a variety, as it is not closed under complementation: we will later see that the language $(\mathtt{a}+\mathtt{b})^*\mathtt{a}(\mathtt{a}+\mathtt{b})^*$ can be maintained, but its complement $\mathtt{b}^*$ cannot. As both languages are recognized by the same monoid $U_1 = \{1, a\}$, the information provided by syntactic monoids alone is not sufficient for understanding $\mathsf{UDyn}\Sigma_1^+$.

Ordered monoids. An algebraic theory for classes of regular languages satisfying all the requirements of a variety but complementation has been developed by Pin [11]. In this theory, syntactic monoids are supplemented by an additional order. An *ordered monoid* is a pair $(M, \leq)$ where M is a monoid and $\leq$ is a partial order on M compatible with the operation, that is, $xx' \leq yy'$ whenever $x \leq y$ and $x' \leq y'$. We write $\uparrow x$ for the set of all elements y with $y \geq x$. We call $P \subseteq M$ an *upset* if it is closed under $\uparrow$, that is, if $\uparrow x \subseteq P$ for all $x \in P$. A *morphism* from $(M, \leq)$ to $(N, \leq)$ is a monoid morphism from M to N such that for all $x, y \in M$, $x \leq y$ implies $\varphi(x) \leq \varphi(y)$. The notions *quotient*, *submonoid* and *division* for monoids can be transferred to ordered monoids using this new notion of morphism.

A language L is *recognized* by $(M, \leq)$ if there is a (monoid) morphism $\varphi : \Sigma^* \to M$ and an upset P such that $L = \varphi^{-1}(P)$. The *syntactic ordered monoid* $(M, \leq)$ for a regular language L consists of the transition monoid M of the minimal automaton and an order $\leq$ such that $\delta_1 \leq \delta_2$ for $\delta_1, \delta_2 \in M$ if for some $\delta_3 \in M$ and any state q of the minimal automaton it holds that if $\delta_3(\delta_1(q))$ is an accepting state, so is $\delta_3(\delta_2(q))$.

▶ **Example 24.** Recall that $U_1 = \{1, a\}$ is the syntactic monoid of $(\mathtt{a}+\mathtt{b})^*\mathtt{a}(\mathtt{a}+\mathtt{b})^*$, and therefore also of its complement $\mathtt{b}^*$. However, these languages can be distinguished by their syntactic ordered monoids. Let U_1^+ (resp. U_1^-) be the ordered monoid U_1 equipped with the order $1 \leq a$ (resp. $1 \geq a$). Then U_1^+ is the syntactic ordered monoid of $(\mathtt{a}+\mathtt{b})^*\mathtt{a}(\mathtt{a}+\mathtt{b})^*$, whereas U_1^- is the syntactic ordered monoid of $\mathtt{b}^*$. Note that $\mathtt{b}^*$ cannot be recognized by U_1^+ because it is the inverse image of 1, which is not an upset.

A prominent example of a class of languages defined via ordered monoids is $\mathbf{J}^+$.

▶ **Definition 25.** *We denote by* $\mathbf{J}^+$ *the class of all languages whose syntactic ordered monoid has the property that* $1 \leq x$ *for every* $x \in M$*.*

It is known[2] [2, Theorem 3.4] that this class coincides with the class of regular languages expressible by Σ_1-formulas. A simple inspection of the proof gives that all formulas are furthermore Σ_1^+-formulas.

[2] In that reference, all orders are reversed. In particular, recognition is defined in terms of *downsets*.

Ordered wreath products. A successful application of wreath products has been to decompose languages into simpler ones, e.g. in the celebrated decomposition theorem of Krohn and Rhodes [7]. One intuition for these products comes from the cascading of finite state automata. Suppose that $\mathcal{A}_1$ is a deterministic finite state automaton over Σ with state set Q, and $\mathcal{A}_2$ is a deterministic finite state automaton over $Q \times \Sigma$. On input w, their cascade product $\mathcal{A}_2 \circ \mathcal{A}_1$ first annotates w with the states reached in its run in $\mathcal{A}_1$. Then the resulting enhanced word is fed into $\mathcal{A}_2$ to check for acceptance.

The algebraic counterpart of this cascade product is the semidirect product of monoids. Here, we use the version for ordered monoids introduced by Pin and Weil [13]. Let $(M, \leq)$ and $(N, \leq)$ be two ordered monoids. To ease notation, we make the common notational convention to denote the operation of M additively.[3] The semidirect product of $(M, \leq)$ and $(N, \leq)$ is defined with respect to a given left action. A *left action* $\cdot$ of $(N, \leq)$ on $(M, \leq)$ is a map $(y, x) \mapsto y \cdot x$ from $N \times M$ to M such that for every $x, x_1x_2 \in M$ and $y, y_1, y_2 \in N$ satisfies the axioms (1) $(y_1y_2) \cdot x = y_1 \cdot (y_2 \cdot x)$, (2) $y \cdot (x_1 + x_2) = y \cdot x_1 + y \cdot x_2$, (3) $1 \cdot x = x$, (4) $y \cdot 1 = 1$, (5) if $x_1 \leq x_2$ then $y \cdot x_1 \leq y \cdot x_2$; and (6) if $y_1 \leq y_2$ then $y_1 \cdot x \leq y_2 \cdot x$.

The *semidirect product* $(M, \leq) * (N, \leq)$ of $(M, \leq)$ and $(N, \leq)$ with respect to the left action $\cdot$ is the ordered monoid on $M \times N$ defined by the operation $(x_1, y_1)(x_2, y_2) = (x_1 + y_1 \cdot x_2, y_1y_2)$ for any $x_1, x_2 \in M$ and $y_1, y_2 \in N$. The order on the product is defined componentwise, that is $(x_1, y_1) \leq (x_2, y_2)$ if and only if $x_1 \leq x_2$ and $y_1 \leq y_2$.

Unfolding the definition for a longer product sheds light on the connection between the semidirect product on monoids and the cascaded product on automata. For $x_1, \ldots, x_n \in M$ and $y_1, \ldots, y_n \in N$, the product in $M * N$ is (by using the axioms defining a left action):

$$(x_1, y_1) \cdots (x_n, y_n) = (x_1 + y_1 \cdot x_2 + (y_1y_2) \cdot x_3 + \cdots + (y_1 \cdots y_{n-1}) \cdot x_n, y_1 \cdots y_n) \qquad (1)$$

Indeed, in a cascade product, the transition function when reading a letter in the first automaton changes according to the transition function realized so far by the second automaton. When reading the i^{th} letter, the annotation is determined by $y_1 \cdots y_{i-1}$ and the action $(y_1 \cdots y_{i-1}) \cdot x_i$ gives the transition function that is used.

The *wreath product* $\mathbf{V} * \mathbf{W}$ of two classes $\mathbf{V}$ and $\mathbf{W}$ of ordered monoids contains all ordered monoids that divide a semidirect product of the form $(M, \leq) * (N, \leq)$ for $(M, \leq) \in \mathbf{V}$ and $(N, \leq) \in \mathbf{W}$.

▶ Remark 26. We chose this definition of wreath products for simplicity. While it already exists in the literature, a most standard choice for $\mathbf{V} * \mathbf{W}$ would have been to take all ordered monoids dividing $M^N \times N$ endowed with some specific order and operation, when $(M, \leq) \in \mathbf{V}$ and $(N, \leq) \in \mathbf{W}$. In fact, both definitions are equivalent when both $\mathbf{V}$ and $\mathbf{W}$ are closed under direct products. This is the case for the classes under study here, that is for $\mathbf{J}^+$ and $\mathbf{G}$. Furthermore, the only result on wreath products that is used out-of-the-shelf is Theorem 4.11 in [13] (stated as Theorem 31 here). Therein, the equivalence between all definitions is proved (see their Proposition 3.5, our definition is (1) and the alternative one is (2)).

5.2 Maintaining regular languages in $\mathbf{J}^+ * \mathbf{G}$ with positive Σ_1-formulas

We will now show that unary auxiliary relations and positive Σ_1-formulas are sufficient to maintain the evaluation problems of ordered monoids in $\mathbf{J}^+ * \mathbf{G}$. The upper bound from Theorem 23 then follows from the following Fact 27 which is the analogon of Fact 6 for

[3] Note that it does not mean that M is commutative.

ordered monoids. For an ordered monoid $(M, \leq)$, define $\textsc{Member}(M, \leq)$ to be the problem that, for a word w subject to changes, maintain a dedicated bit q_x that holds 1 whenever $w \in \uparrow x$, for every $x \in M$.

▶ **Fact 27.** *Let $\mathcal{C}$ be a class of formulas closed under $\vee$. Further let L be a regular language and $(M, \leq)$ its syntactic ordered monoid. Then:*

$$\textsc{Member}(M, \leq) \in \mathsf{UDyn}\mathcal{C} \Rightarrow \textsc{Member}(L) \in \mathsf{UDyn}\mathcal{C}.$$

▶ **Lemma 28.** *If $(M, \leq) \in \mathbf{J}^+ * \mathbf{G}$, then $\textsc{Member}(M, \leq)$ is in $\mathsf{UDyn}\Sigma_1^+$.*

Proof sketch. We show how to maintain $\textsc{Member}(M, \leq)$ when $(M, \leq) = (J, \leq) * (G, \leq)$ for a fixed left action, where $(J, \leq) \in \mathbf{J}^+$ and G is a group. The result then follows from Fact 7, that can be easily extended to ordered monoids.

Let $w = (x_1, g_1) \cdots (x_n, g_n)$ be the input word, where $x_i \in J$ and $g_i \in G$ for all $i \leq n$. Let $P = \uparrow (p, q) = (\uparrow p, \uparrow q)$ be an upset of $(M, \leq)$, where $p \in J$ and $q \in G$. We describe how to maintain whether w evaluates in P.

By Lemma 16, we can maintain all evaluations $g_1 \cdots g_{i-1}$ of strict prefixes of the projection of w to the component from G in $\mathsf{UDynProp}$, as well as the information whether $g_1 \cdots g_n$ evaluates in $\uparrow q$. The corresponding update formulas are actually positive: the original update formulas in [4] are positive and the adaptations of the proof of Lemma 16 do not introduce negations. This allows to maintain the values $(g_1 \cdots g_{i-1}) \cdot x_i \in J$ using positive quantifier-free formulas.

Using the Equation 1, we still need to determine whether $x_1 + g_1 \cdot x_2 + (g_1 g_2) \cdot x_3 + \cdots + (g_1 \cdots g_{n-1}) \cdot x_n$ evaluates in $\uparrow p$. As the membership problem of $(J, \leq)$ can be expressed by a Σ_1^+ formula [2, Theorem 3], Σ_1^+ update formulas can perform that evaluation. ◀

5.3 Lower bound for regular languages not in $\mathbf{J}^+ * \mathbf{G}$

We want to show that Lemma 28 is optimal and any language whose ordered syntactic monoid is not in $\mathbf{J}^+ * \mathbf{G}$ cannot be maintained in $\mathsf{UDyn}\Sigma_1^+$. The proof is in two steps. First, we show that the regular language $\mathtt{b}^*$ cannot be maintained in $\mathsf{UDyn}\Sigma_1^+$. We then introduce a characterization of $\mathbf{J}^+ * \mathbf{G}$ that implies that if a language not recognized by $\mathbf{J}^+ * \mathbf{G}$ could be maintained, then so could the language $\mathtt{b}^*$.

5.3.1 A regular language that is not in $\mathsf{UDyn}\Sigma_1^+$

We show that the language b^* is not in $\mathsf{UDyn}\Sigma_1^+$ by generalizing the substructure lemma from $\mathsf{DynProp}$ to $\mathsf{UDyn}\Sigma_1^+$ and then applying it to b^*.

The substructure lemma for $\mathsf{UDynProp}$ relies on the inability of quantifier-free formulas to access elements outside the isomorphic substructures. We adopt the lemma to allow existential quantification in update formulas by weakening the requirement on the substructures to be isomorphic. To this end, let $(w, \mathcal{A}), (w', \mathcal{A}')$ be words of length $n \leq m$ respectively annotated with sets of unary auxiliary relations $\mathcal{A}, \mathcal{A}'$. For $i \in [n]$, the *type* of $(w, \mathcal{A})_{|i}$ consists of w_i and of the set of all relations of $\mathcal{A}$ containing i. Let π be a mapping from $[n]$ to $[m]$. We say that π is *type-monotonic* if the type of $(w, \mathcal{A})_{|i}$ is a subset of the type of $(w', \mathcal{A}')_{|\pi(i)}$ for all $i \in [n]$, i.e. if whenever $R^{\mathcal{A}}(i)$ holds then so does $R^{\mathcal{A}'}(\pi(i))$. In this case, we say that $(w, \mathcal{A})$ and $(w', \mathcal{A}')_{|\pi([n])}$ are type-monotonic via π.

▶ **Lemma 29** (Substructure lemma for $\mathsf{UDyn}\Sigma_1^+$)**.** *Let Π be a dynamic $\mathsf{UDyn}\Sigma_1^+$-program and let $w \in \Sigma^n$ and $w' \in \Sigma^m$ be two words over the alphabet Σ. Further, let $\mathcal{A}$ and $\mathcal{A}'$ be sets of at most unary auxiliary relations over the domain of w and w' and the schema of Π.*

Assume there is $I \subseteq [m]$ such that $(w, \mathcal{A})$ and $(w', \mathcal{A}')_{|I}$ are type-monotonic via some order-preserving mapping π. Then π is also a type-monotonic mapping from $\Pi_\alpha(w, \mathcal{A})$ to $\Pi_{\alpha'}(w', \mathcal{A}')_{|I}$, for any π-respecting sequences α and α' of changes.

Proof. We show the statement for a single change $\alpha = (\text{SET}_\sigma(i))$ and $\alpha' = (\text{SET}_\sigma(\pi(i))$ with $i \in [n]$; the general statement follows by induction on the length of change sequences.

For showing that $\Pi_\alpha(w, \mathcal{A})$ and $\Pi_{\alpha'}(w', \mathcal{A}')_{|I}$ are type-monotonic via π, consider a unary relation symbol R updated by formula $\varphi(x; y) \stackrel{\text{def}}{=} \exists z_1 \dots \exists z_k \psi \in \Sigma_1^+$ when a change to σ is made. Suppose $\varphi(j; i)$ evaluates to true in $(w, \mathcal{A})$ by choosing $z_1, \dots, z_k$ as $j_1, \dots, j_k$. Then $\varphi(\pi(j); \pi(i))$ evaluates to true in $(w', \mathcal{A}')$ by choosing $z_1, \dots, z_k$ as $\pi(j_1), \dots, \pi(j_k)$, because $(w, \mathcal{A})$ and $(w', \mathcal{A}')_{|I}$ are type-monotonic via π and ψ only contains positive atoms. ◀

To conclude, we apply Higman's lemma in a similar fashion as in Section 4.

▶ **Lemma 30.** *The language $\mathtt{b}^*$ is not in $\mathsf{UDyn}\Sigma_1^+$.*

Proof. Assume towards a contradiction that there is a dynamic $\mathsf{UDyn}\Sigma_1^+$-program Π that maintains $\mathtt{b}^*$. We consider the sequence $(w_i, \mathcal{A}_i)_{i \geq 1}$, where w_i is the word $\mathtt{a}^i$ and $\mathcal{A}_i$ are the auxiliary relations that Π obtains when starting from an empty input word of size i and setting all positions to $\mathtt{a}$. By Lemma 18, there are $n < m$, $I \subseteq [m]$ and π such that π is an order-preserving isomorphism from $(w_n, \mathcal{A}_n)$ to $(w_m, \mathcal{A}_m)_{|I}$. This implies, in particular, that π is type-monotonic on $(w_n, \mathcal{A}_n)$ and $(w_m, \mathcal{A}_m)_{|I}$.

The sequences $\alpha = \text{SET}_b(1) \cdots \text{SET}_b(n)$ and $\alpha' = \text{SET}_b(\pi(1)) \cdots \text{SET}_b(\pi(n))$ are π respecting, so according to Lemma 29, the structures $\Pi_\alpha(w_n, \mathcal{A}_n)$ and $\Pi_{\alpha'}(w_m, \mathcal{A}_m)_{|I}$ are also type-monotonic. But then, because the first word belongs to the language after applying the changes, the program must also say so for the second word after applying the changes (as it must be monotonic on the answer bit). This is a contradiction, because the second word does not belong to the language after the changes. ◀

5.3.2 $\mathsf{UDyn}\Sigma_1^+$ cannot maintain any regular non-$\mathbf{J}^+ * \mathbf{G}$ language

To lift the unmaintainability result of Lemma 30 to all regular languages whose syntactic monoid is not in $\mathbf{J}^+ * \mathbf{G}$, we first show that $\mathsf{UDyn}\Sigma_1^+$ cannot maintain the membership problem of such monoids and then lift this to all languages recognized by them.

We have seen that the definition of $\mathbf{J}^+ * \mathbf{G}$ with the wreath product is useful to design algorithms using only formulas of low complexity and to obtain upper bounds. However, for understanding which monoids do not belong to the class, an equivalent characterization due to Pin and Weil [13] is helpful. Denote by $\mathbf{EJ}^+$ the class[4] of languages whose syntactic ordered monoids satisfy $1 \leq e$ for every idempotent e. For instance, $(\mathtt{a} + \mathtt{b})^*\mathtt{aa}(\mathtt{a} + \mathtt{b})^*$ is not in $\mathbf{J}^+$ but in $\mathbf{EJ}^+$, while $(\mathtt{ab})^*$ is in neither. Furthermore, every group is in $\mathbf{EJ}^+$ as they have a single idempotent which is the identity.

▶ **Theorem 31** (Pin and Weil [13, Thm 4.11])**.** $\mathbf{J}^+ * \mathbf{G} = \mathbf{EJ}^+$

This characterization immediately yields U_1^- as witness for non-membership in $\mathbf{J}^+ * \mathbf{G}$.

▶ **Lemma 32.** *If $(M, \leq) \notin \mathbf{J}^+ * \mathbf{G}$, then U_1^- divides $(M, \leq)$.*

Proof. By definition of $\mathbf{EJ}^+$, we can find an idempotent $e \in M$ such that $1 \not\leq e$. It implies in particular that e is different from the identity. We consider the submonoid $(U_1, \leq)$ of $(M, \leq)$ with $U_1 = \{1, e\}$ and do a case distinction on $\leq$:

[4] Note Pin and Weil denote $\mathbf{EJ}^+$ by $\mathbf{BG}^+$ and reverse the order.

- If $1 \geq e$, then $(U_1, \leq)$ is exactly U_1^-.
- If 1 and e are incomparable, then the order $\leq$ restricted to U_1 is the equality $=$. We conclude by remarking that U_1^- is a quotient of $(U_1, =)$, using the identity function. ◀

▶ **Lemma 33.** *If $(M, \leq)$ is an ordered monoid that is not in $\mathbf{J}^+ * \mathbf{G}$, then* Member$(M, \leq)$ *is not in* UDynΣ_1^+.

Proof. We first observe that UDynΣ_1^+ cannot maintain Member(U_1^-), due to Fact 27 and Lemma 19 and because U_1^- is the ordered syntactic monoid of the language $\mathfrak{b}^*$.

Now assume, towards a contradiction, that Member$(M, <)$ is in UDynΣ_1^+ for some monoid $(M, <)$ which is not in $\mathbf{J}^+ * \mathbf{G}$. By Lemma 32, U_1^- is a submonoid of $(M, \leq)$ and hence Member(U_1^-) is in UDynΣ_1^+ by Fact 7 straightforwardly extended to ordered monoids (it suffices to remark that the inverse image of an upset by a morphism is an upset), contradicting our observation from above. ◀

Now we have to translate the previous lemma from ordered monoids to languages, that is, we need a converse to Fact 27. To this end, we prove that if UDynΣ_1^+ can maintain a regular language, it can also maintain the membership problem of its ordered syntactic monoid.

To this end, we use a variant of varieties that does not require closure under complementation. A class $\mathcal{V}$ of regular languages is a *positive (pseudo)variety* if it is closed under positive Boolean operations, quotients and inverse morphisms.

Examples of positive varieties are $\mathbf{V} * \mathbf{W}$ whenever $\mathbf{V}$ and $\mathbf{W}$ are positive varieties [13, Proposition 3.5]. Also UDynΣ_1^+ is a positive variety, because it is closed under positive Boolean operations (because Σ_1^+ is closed under $\vee$ and $\wedge$) and under quotients and inverse morphisms (due to Lemma 5). The theorem of Eilenberg translates to positive varieties:

▶ **Theorem 34** (Pin [12, Theorem 4.12])**.** *Let $\mathcal{V}$ be a positive variety of regular languages and let $(M, \leq)$ be the syntactic ordered monoid of a language $L \in \mathcal{V}$. Then any language recognized by $(M, \leq)$ is also in $\mathcal{V}$.*

This implies that the converse of Fact 27 holds for positive varieties.

▶ **Fact 35.** *Let $\mathcal{C}$ be a class of formulas such that the regular languages of* UDyn$\mathcal{C}$ *form a positive variety. Let L be a regular language and $(M, \leq)$ its syntactic ordered monoid:*

$$\text{Member}(L) \in \text{UDyn}\mathcal{C} \Rightarrow \text{Member}(M, \leq) \in \text{UDyn}\mathcal{C}.$$

Proof. Assume that Member(L) is in UDyn$\mathcal{C}$. Let $\varphi : M^* \to M$ be the morphism that evaluates a word in M^*, that is that maps $x_1 \cdots x_n$ to the product of the x_i for $1 \leq i \leq n$. For any fixed $x \in M$, notice that $\uparrow x$ is an upset of M. Thus Member$(\varphi^{-1}(\uparrow x))$ is in UDyn$\mathcal{C}$ for every $x \in M$, by Theorem 34. A dynamic program for Member(M) uses each of these programs to generate one bit of output. ◀

The "only if" direction of Theorem 23 now follows. By Fact 35 and because the regular languages of UDynΣ_1^+ form a positive variety: if UDynΣ_1^+ can maintain a regular language, it can also maintain the membership problem of its ordered syntactic monoid. But, by Lemma 33, UDynΣ_1^+ cannot maintain this problem for any ordered monoid not in $\mathbf{J}^+ * \mathbf{G}$.

6 Conclusion

We characterized the regular languages of UDynΣ_1^+ and UDynProp by properties of their (ordered) syntactic monoids. The most important fragment of DynFO for which a characterization remains open is UDynΣ_1 (see also the discussion in the full version of this paper).

▶ **Open question.** Which regular languages are in $\mathsf{UDyn}\Sigma_1$, i.e. can be maintained by $\exists^*$ update formulas and unary auxiliary relations?

Two obstacles to an algebraic characterization of $\mathsf{UDyn}\Sigma_1$ are (1) that Σ_1 is not closed under composition and thus $\mathsf{UDyn}\Sigma_1$ is a priori not a (positive) variety; and (2) an absence of lower bounds techniques against $\mathsf{UDyn}\Sigma_1$. One path towards resolving (1) is via considering positive lp-varieties, which only require closure under positive Boolean operations and under quotients of inverses of length-preserving morphisms [14]. For (2), new lower bound techniques beyond the substructure lemma (and its variant used for $\mathsf{UDyn}\Sigma_1^+$) seem to be necessary.

We shortly discuss the expressive power of $\mathsf{UDyn}\Sigma_1$. Consider the class $\mathbf{J}$ of languages that can be described by a Boolean combination of Σ_1-formulas. This class is equal to the class of piecewise-testable languages [18]. We can show that $\mathsf{UDyn}\Sigma_1$ is strictly more powerful than $\mathsf{UDyn}\Sigma_1^+$, by looking at the unordered counterpart of $\mathbf{J}^+ * \mathbf{G}$.

▶ **Lemma 36.** *If L is a regular language in $\mathbf{J} * \mathbf{G}$, then* $\textsc{Member}(L)$ *is in* $\mathsf{UDyn}\Sigma_1$.

Unfortunately, $\mathbf{J} * \mathbf{G}$ does not provide an exact characterization, because the language $\Sigma^*\mathtt{aa}\Sigma^*$ is in $\mathsf{UDyn}\Sigma_1$, but its ordered syntactic monoid is not in $\mathbf{J} * \mathbf{G}$.

▶ **Conjecture 37.** *Let $\mathcal{V}$ be the positive lp-variety of regular languages L such that* $\textsc{Member}(L)$ *is in* $\mathsf{UDyn}\Sigma_1$. *Then $\mathbf{J} * \mathbf{G} \subsetneq \mathcal{V} \subseteq \Sigma_2 * \mathbf{G}$. In particular, $(\Sigma^*\mathtt{aa}\Sigma^*)^\mathtt{c}$ is not*[5] *in $\mathcal{V}$.*

Another direction is to explore (fragments of) the class of context-free languages. It is known that every context-free language can be maintained in DynFO [4], yet the known dynamic program requires Σ_1-update formulas and 4-ary auxiliary relations.

▶ **Open question.** Which dynamic resources (quantifier-alterations, arity of auxiliary relations,...) are required for maintaining context-free languages?

References

1 J. Büchi. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6:66–92, 1960. doi:10.1007/978-1-4613-8928-6_22.

2 Volker Diekert, Paul Gastin, and Manfred Kufleitner. A survey on small fragments of first-order logic over finite words. *Int. J. Found. Comput. Sci.*, 19(3):513–548, 2008. doi:10.1142/S0129054108005802.

3 Samuel Eilenberg. *Automata, Languages and Machines, Vol. B.* Academic Press Inc, 1976. URL: https://www.sciencedirect.com/bookseries/pure-and-applied-mathematics/vol/59/part/PB.

4 Wouter Gelade, Marcel Marquardt, and Thomas Schwentick. The dynamic complexity of formal languages. *ACM Trans. Comput. Logic*, 13(3), 2012. doi:10.1145/2287718.2287719.

5 J. A. Green. On the structure of semigroups. *Annals of Mathematics*, 54(1):163–172, 1951. URL: http://www.jstor.org/stable/1969317.

6 William Hesse. *Dynamic Computational Complexity.* PhD thesis, University of Massachusetts Amherst, 2003.

7 Kenneth Krohn and John Rhodes. Algebraic theory of machines. i. prime decomposition theorem for finite semigroups and machines. *Transactions of the American Mathematical Society*, 116:450–464, 1965. URL: http://www.jstor.org/stable/1994127.

8 M. Lothaire. *Combinatorics on Words.* Cambridge Mathematical Library. Cambridge University Press, 2 edition, 1997.

[5] Note that non-membership in $\Sigma_2 * \mathbf{G}$ can be checked with the MeSCaL software [15].

9 Robert McNaughton and Seymour A. Papert. *Counter-Free Automata (M.I.T. research monograph no. 65)*. The MIT Press, 1971.

10 Sushant Patnaik and Neil Immerman. Dyn-FO: A parallel, dynamic complexity class. *J. Comput. Syst. Sci.*, 55(2):199–209, 1997. doi:10.1006/jcss.1997.1520.

11 Jean-Éric Pin. A variety theorem without complementation. *Russian Mathematics (Izvestija vuzov. Matematika)*, 39:80–90, 1995.

12 Jean-Eric Pin. Mathematical foundations of automata theory. Lecture notes, 2014. URL: http://www.irif.fr/~jep/PDF/MPRI/MPRI.pdf.

13 Jean-Eric Pin and Pascal Weil. Semidirect products of ordered semigroups. *Communications in Algebra*, 30(1):149–169, 2002.

14 Jean-Éric Pin and Howard Straubing. Some results on c-varieties. *RAIRO - Theoretical Informatics and Applications*, 39(1):239–262, March 2010. URL: http://eudml.org/doc/92759.

15 Thomas Place and Marc Zeitoun. A first taste of mescal, a tool for solving membership problems for regular languages. In Giuseppa Castiglione and Sabrina Mantaci, editors, *Implementation and Application of Automata - 29th International Conference, CIAA 2025, Palermo, Italy, September 22-25, 2025, Proceedings*, volume 15981 of *Lecture Notes in Computer Science*, pages 281–298. Springer, 2025. doi:10.1007/978-3-032-02602-6_20.

16 M. P. Schützenberger. Une théorie algébrique du codage. *Séminaire Dubreil. Algèbre et théorie des nombres*, 9:1–24, 1955. URL: http://eudml.org/doc/111094.

17 M.P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965. doi:10.1016/S0019-9958(65)90108-7.

18 Imre Simon. Piecewise testable events. In H. Brakhage, editor, *Automata Theory and Formal Languages*, pages 214–222, Berlin, Heidelberg, 1975. Springer Berlin Heidelberg. doi:10.1007/3-540-07407-4_23.

19 Gudmund Skovbjerg Frandsen, Peter Bro Miltersen, and Sven Skyum. Dynamic word problems. *J. ACM*, 44(2):257–271, 1997. doi:10.1145/256303.256309.

20 Thomas Zeume. *Small dynamic complexity classes*. PhD thesis, Technical University Dortmund, Germany, 2015. URL: https://hdl.handle.net/2003/34163.

21 Thomas Zeume and Thomas Schwentick. On the quantifier-free dynamic complexity of reachability. *Information and Computation*, 240:108–129, 2015. MFCS 2013. doi:10.1016/j.ic.2014.09.011.