The Intersection Problem for Finite Monoids

Lukas Fleischer
FMI, University of Stuttgart,
Universitätsstraße 38, 70569 Stuttgart, Germany
fleischer@fmi.uni-stuttgart.de

Manfred Kufleitner
Department of Computer Science, Loughborough University,
Epinal Way, Loughborough LE11 3TU, United Kingdom
m.kufleitner@lboro.ac.uk

Abstract

We investigate the intersection problem for finite monoids, which asks for a given set of regular languages, represented by recognizing morphisms to finite monoids from a variety $\mathbf{V}$, whether there exists a word contained in their intersection. Our main result is that the problem is PSPACE-complete if $\mathbf{V} \not\subseteq \mathbf{DS}$ and NP-complete if $\mathbf{1} \subsetneq \mathbf{V} \subseteq \mathbf{DO}$. Our NP-algorithm for the case $\mathbf{V} \subseteq \mathbf{DO}$ uses novel methods, based on compression techniques and combinatorial properties of $\mathbf{DO}$. We also show that the problem is log-space reducible to the intersection problem for deterministic finite automata (DFA) and that a variant of the problem is log-space reducible to the membership problem for transformation monoids. In light of these reductions, our hardness results can be seen as a generalization of both a classical result by Kozen and a theorem by Beaudry, McKenzie and Thérien.

2012 ACM Subject Classification Theory of computation → Regular languages, Theory of computation → Algebraic language theory, Theory of computation → Problems, reductions and completeness

Keywords and phrases Intersection Problem, Finite Monoid, Recognizing Morphism, Complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2018.30

Funding This work was supported by the DFG grants DI 435/5-2 and KU 2716/1-1.

1 Introduction

In 1977, Kozen showed that deciding whether the intersection of the languages recognized by a set of given deterministic finite automata (DFA) is non-empty is PSPACE-complete [16]. This result has since been the building block for numerous hardness results in formal language theory and related fields; see e.g. [8, 11, 12, 15]. It is natural to ask whether the problem becomes easier when restricting the input. Various special cases, such as bounding the number k of automata in the input [17] or considering only automata with a fixed number of accepting states [9], were investigated in follow-up work; see [14] for a survey.

Another very natural restriction is to only consider automata with certain *structural properties*. One such property is *counter-freeness*: an automaton is *counter-free* if no word permutes a non-trivial subset of its states. By a famous result of Schützenberger [19], a regular language is recognized by a counter-free automaton if and only if it is star-free. These properties are often expressed using the algebraic framework: instead of considering the automaton itself, one considers its transition monoid. The latter is the transformation monoid generated by the action of the letters on the set of states. Now, properties of automata

Table 1 Summary of complexity results (new main result, follows from reductions.)

	MonIsect(**V**)	MonIsect$_1$(**V**)	Memb(**V**) [2, 7]
NC	—	$\mathbf{V} \subseteq \mathbf{G}$	$\mathbf{V} \subseteq \mathbf{G}$
P	—	$\mathbf{V} \subseteq \mathbf{R_1} \vee \mathbf{L_1}$	$\mathbf{V} \subseteq \mathbf{R_1} \vee \mathbf{L_1}$
NP	$\mathbf{V} \subseteq \mathbf{DO}$	$\mathbf{V} \subseteq \mathbf{DO}$	$\mathbf{V} \subseteq \mathbf{R}, \mathbf{V} \subseteq \mathbf{A_1}$
NP-hard	all $\mathbf{V} \neq \mathbf{1}$	—	$\mathbf{Acom_2} \subseteq \mathbf{V}$, $\mathbf{XR} \subseteq \mathbf{V}, \mathbf{XL} \subseteq \mathbf{V}$
PSPACE	all **V**	all **V**	all **V**
PSPACE-hard	$\mathbf{V} \not\subseteq \mathbf{DS}$	$\mathbf{V} \not\subseteq \mathbf{DS}$	$\mathbf{V} \not\subseteq \mathbf{DS}$

can be given by membership of the transition monoid in certain classes, so-called *varieties*, of finite monoids. For example, an automaton is counter-free if and only if its transition monoids belongs to the variety **A** of *aperiodic* monoids. The DFA intersection problem for a variety **V**, denoted by DfaIsect(**V**), is formalized as follows.

DfaIsect(**V**)	
Input:	DFAs $A_1, \ldots, A_k$ with transition monoids from **V**
Question:	Is $L(A_1) \cap \cdots \cap L(A_k) \neq \emptyset$?

Note that DfaIsect(**Mon**), where **Mon** is the variety of all finite monoids, is the general DFA intersection problem considered by Kozen. A careful inspection of his proof actually reveals that DfaIsect(**A**) is PSPACE-complete already [11]. Additionally requiring all DFAs to have a single accepting state, we obtain a variant of DfaIsect(**V**) reminiscent of another problem investigated by Kozen, the *membership problem for transformation monoids*.

Memb(**V**)	
Input:	Transformations $f_1, \ldots, f_m \colon X \to X$ generating a monoid $T \in \mathbf{V}$ and $g \colon X \to X$
Question:	Does g belong to T?

The complexity of Memb(**V**) was studied extensively in a series of papers [2, 4, 5, 6, 7, 13, 3]. However, for certain varieties **V**, obtaining the exact complexity of DfaIsect(**V**) and Memb(**V**) is a challenging problem. To date, only partial results are known, see Table 1. For example, it is open whether or not Memb(**DA**) $\in$ NP, a question stated explicitly in [7] and revisited in [20] around ten years later.

Since algebraic tools are already used to express structural properties of automata, it seems natural to consider the fully algebraic version of the intersection problem by directly using finite monoids as language acceptors instead of taking the detour via automata and their transition monoids. A language $L \subseteq A^*$ is *recognized* by a morphism $h \colon A^* \to M$ to a finite monoid M if $L = h^{-1}(P)$ for some subset P of M. The set P is often called the *accepting set* because it resembles the accepting states in finite automata. A monoid M recognizes a language $L \subseteq A^*$ if there exists a morphism $h \colon A^* \to M$ recognizing L. It is well-known that a language is recognized by a finite monoid if and only if it is regular. For a variety of finite monoids **V**, the *intersection problem for* **V** is defined as follows.

MonIsect(**V**)	
Input:	Morphisms $h_i \colon A^* \to M_i \in \mathbf{V}$ and sets $P_i \subseteq M_i$ with $1 \leqslant i \leqslant k$
Question:	Is $h_1^{-1}(P_1) \cap \cdots \cap h_k^{-1}(P_k) \neq \emptyset$?

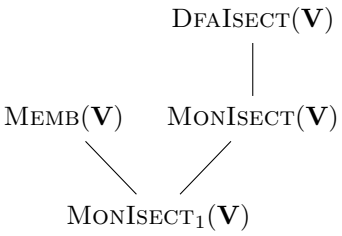

Figure 1 Relations between the problems considered in this work.

We assume that the monoids are given as multiplication tables, such that, assuming a random-access machine model, multiplications can be performed in logarithmic time.

There is a close connection to both the DFA intersection problem and the membership problem for transformation monoids. More specifically, for every variety $\mathbf{V}$, there is a log-space reduction of $\textsc{MonIsect}(\mathbf{V})$ to $\textsc{DfaIsect}(\mathbf{V})$. The variant $\textsc{MonIsect}_1(\mathbf{V})$ of the finite monoid intersection problem, where each of the accepting sets is a singleton, can be reduced to $\textsc{Memb}(\mathbf{V})$. Our reducibility results are depicted in Figure 1.

Not only is the algebraic version of the intersection problem a natural problem to consider, making progress in classifying its complexity also raises hope to make progress in solving open complexity questions regarding $\textsc{DfaIsect}(\mathbf{V})$ and $\textsc{Memb}(\mathbf{V})$. Using novel techniques, we prove that $\textsc{MonIsect}(\mathbf{V})$ is NP-complete whenever $\mathbf{V} \subseteq \mathbf{DO}$ and PSPACE-complete whenever $\mathbf{V} \not\subseteq \mathbf{DS}$. In particular, since $\mathbf{DA}$ is a subset of $\mathbf{DO}$, we obtain an NP-algorithm for $\textsc{MonIsect}(\mathbf{DA})$ while the problem of whether there exists such an algorithm for $\textsc{Memb}(\mathbf{DA})$ or $\textsc{DfaIsect}(\mathbf{DA})$ has been open for more than 25 years. Moreover, in view of the reductions mentioned above, our PSPACE-hardness result can be seen as a generalization of both Kozen's result and a result from [7], stating that every variety of aperiodic monoids not contained within $\mathbf{DA} = \mathbf{DS} \cap \mathbf{A}$ admits a PSPACE-complete transformation monoid membership problem.

Our results are summarized in Table 1. Only a very small gap of varieties contained within $\mathbf{DS}$ but not $\mathbf{DO}$ remains. Answering complexity questions in this setting is deeply connected to understanding the languages recognized by monoids in $\mathbf{DS}$ which is another problem open for over twenty years; see e.g. [1, Open Problem 14]. Obtaining a dichotomy result for $\textsc{MonIsect}(\mathbf{V})$ is likely to provide new major insights for both $\textsc{DfaIsect}(\mathbf{V})$ and the language variety corresponding to $\mathbf{DS}$, and, conversely, new insights on either language properties of $\mathbf{DS}$ or on $\textsc{DfaIsect}(\mathbf{DS})$ will potentially help with obtaining such a result.

We conclude with a first complexity result on the intersection problem for finite monoids.

▶ **Theorem 1.** $\textsc{MonIsect}(\mathbf{Mon}) \in$ PSPACE.

Proof. Since PSPACE = NPSPACE by Savitch's Theorem, it suffices to give a non-deterministic algorithm which requires polynomial space. The algorithm proceeds by guessing a word in the intersection, letter by letter. The word is not written down explicitly but after each guess, the image of the current prefix under each morphism is computed and stored. Finally, the algorithm verifies that each of the images is in the corresponding accepting set. ◀

2 Preliminaries

Words and Languages. Let A be a finite alphabet. A *word* over A is a finite sequence of letters $a_1 \cdots a_\ell$ with $a_i \in A$ for all $i \in \{1, \ldots, \ell\}$. The set A^* denotes the set of all words over A and a *language* is a subset of A^*. The *content* (or *alphabet*) of a word $w = a_1 \cdots a_\ell$

is the subset $\mathsf{alph}(w) = \{a_1, \ldots, a_\ell\}$ of A. A word u is a *factor* of w if there exist $p, q \in A^*$ such that $w = puq$; and, when the factorization is fixed, then the position of u is called its *occurrence*.

Algebra. Let M be a finite monoid. An element $e \in M$ is *idempotent* if $e^2 = e$. The set of all idempotent elements of M is denoted by $E(M)$. In a finite monoid M, the integer $\omega_M = |M|!$ plays an important role: for each $m \in M$, the element m^{ω_M} is idempotent. For convenience, we often write ω instead of ω_M if the reference to M is clear from the context. For two elements $m, n \in M$, we write $m \leqslant_{\mathcal{J}} n$ if the two-sided ideal of m is contained in the two-sided ideal of n, i.e., $MmM \subseteq MnM$. We write $m \mathcal{J} n$ if both $m \leqslant_{\mathcal{J}} n$ and $n \leqslant_{\mathcal{J}} m$.

The *direct product* of two monoids M and N is the Cartesian product $M \times N$ with componentwise multiplication. A monoid N is a *quotient* of a monoid M if there exists a surjective morphism $h\colon M \to N$. A monoid N is a *divisor* of a monoid M if N is a quotient of a submonoid of M.

A *variety* of finite monoids is a class $\mathbf{V}$ of finite monoids which is closed under (finite) direct products and divisors. The class of all finite monoids **Mon** is a variety. The following other varieties play an important role in this work:

$$\mathbf{G} = \{M \in \mathbf{Mon} \mid \forall e \in E(M) : e = 1\}$$
$$\mathbf{DS} = \{M \in \mathbf{Mon} \mid \forall e, f \in E(M) : e \mathcal{J} f \implies (efe)^\omega = e\}$$
$$\mathbf{DO} = \{M \in \mathbf{Mon} \mid \forall e, f \in E(M) : e \mathcal{J} f \implies efe = e\}$$

It is easy to see that $\mathbf{G}$ contains exactly those finite monoids which are groups. Since direct products of groups are groups and divisors of groups are groups, $\mathbf{G}$ is indeed a variety. For proofs that $\mathbf{DS}$ and $\mathbf{DO}$ are varieties, we refer to [18]. From the definitions, it follows immediately that $\mathbf{DO} \subseteq \mathbf{DS}$. There exist several other interesting characterizations of $\mathbf{DS}$. Let B_2^1 be the monoid defined on the set $\{1, a, b, ab, ba, 0\}$ by the operation $aba = a$, $bab = b$ and $a^2 = b^2 = 0$ where 0 is a zero element. Then the following holds, see e.g. [1].

▶ **Proposition 2.** *Let M be a finite monoid. The following properties are equivalent:*
1. *$M \in \mathbf{DS}$.*
2. *For each $e \in E(M)$ and $x \in M$ with $e \leqslant_{\mathcal{J}} x$, we have $(exe)^\omega = e$.*
3. *For each $e \in E(M)$, the elements $\{x \in M \mid e \leqslant_{\mathcal{J}} x\}$ form a submonoid of M.*
4. *B_2^1 is not a divisor of $M \times M$.*

Tiling Systems. A *tiling system* is a tuple $\mathcal{T} = (\Lambda, T, n, f, b)$ where Λ is a finite set of *labels*, $T \subseteq \Lambda \times \Lambda \times \Lambda \times \Lambda$ are the so-called *tiles*, $n \in \mathbb{N}$ is the *width* and $f, b \in T^n$ are the *first row* and *bottom row*. For a tile $t = (t_w, t_e, t_s, t_n) \in T$, we let $\lambda_w(t) = t_w$, $\lambda_e(t) = t_e$, $\lambda_s(t) = t_s$ and $\lambda_n(t) = t_n$. These labels can be thought of as labels in *west*, *east*, *south* and *north* direction. An *m-tiling* of $\mathcal{T}$ is a mapping $\tau\colon \{1, \ldots, m\} \times \{1, \ldots, n\} \to T$ such that the following properties hold:
1. $\tau(1,1)\tau(1,2)\cdots\tau(1,n) = f$,
2. $\lambda_e(\tau(i,j)) = \lambda_w(\tau(i,j+1))$ for $1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant n-1$,
3. $\lambda_s(\tau(i,j)) = \lambda_n(\tau(i+1,j))$ for $1 \leqslant i \leqslant m-1$ and $1 \leqslant j \leqslant n$,
4. $\tau(m,1)\tau(m,2)\cdots\tau(m,n) = b$.

The *corridor tiling problem* asks for a given tiling system $\mathcal{T}$ whether there exists some $m \in \mathbb{N}$ such that there is a m-tiling of $\mathcal{T}$. The *square tiling problem* asks for a given tiling system $\mathcal{T}$ of width n, whether there exists an n-tiling of $\mathcal{T}$. It is well-known that the corridor tiling problem is PSPACE-complete and that the square tiling problem is NP-complete [10].

Straight-Line Programs. A *straight-line program* (*SLP*) is a grammar $S = (V, A, P, X_s)$ where V is a finite set of variables, A is a finite alphabet, $P\colon V \to (V \cup A)^*$ is a mapping and $X_s \in V$ is the so-called *start variable*. For a variable $X \in V$, the word $P(X)$ is the *right-hand side* of X. We require that there exists a linear order $<$ on V such that $Y < X$ whenever $P(X) \in (V \cup A)^* Y (V \cup A)^*$. Starting with some word $\alpha \in (V \cup A)^*$ and repeatedly replacing variables $X \in V$ by $P(X)$ yields a word from A^*, the so called *evaluation of* α, denoted by $\mathsf{val}(\alpha)$. The word *produced by* S is $\mathsf{val}(S) = \mathsf{val}(X_s)$. If the reference to A and V is clear from the context, we will often use the notation $h(\alpha)$ instead of $h(\mathsf{val}(\alpha))$ for the image of the evaluation of a word $\alpha \in (A \cup V)^*$ under a morphism $h\colon A^* \to M$. Analogously, we write $h(S)$ instead of $h(\mathsf{val}(S))$. The *size* of S is $|S| = \sum_{X \in V} |P(X)|$. Each variable X of an SLP S can be viewed as an SLP itself by making X the start variable of S.

The following simple lemma illustrates how SLPs can be used for compression.

▶ **Lemma 3.** *Let $S = (V, A, P, X_s)$ be an SLP and let $e \in \mathbb{N}$. Let w be the word produced by S. Then there exists an SLP S' of size $|S'| \leqslant |S| + 4\log(e)$ such that S' produces w^e.*

Proof. We obtain S' by iteratively adding new variables to V as follows, starting with $i = e$ and repeating the process until $i = 0$.

- If $i > 0$ is odd, add a new variable X_i and let $P(X_i) = X_{i-1}X_s$. Let $i := i - 1$.
- If $i > 0$ is even, add a new variable X_i and let $P(X_i) = X_{i/2}X_{i/2}$. Let $i := i/2$.

Finally, add the variable X_0 and let $P(X_0) = \varepsilon$. The new start variable is X_e and by construction, we have $\mathsf{val}(X_e) = w^e$. ◀

3 Connections to Other Problems

Before investigating the complexity of MonIsect($\mathbf{V}$) itself, we establish connections to other well-known problems defined in the introduction, starting with the DFA intersection problem.

▶ **Proposition 4.** *Let $\mathbf{V}$ be a variety of finite monoids, let $M \in \mathbf{V}$, let $h\colon A^* \to M$ be a morphism and let $P \subseteq M$. Then there exists a finite deterministic automaton A with $|M|$ states such that $L(A) = h^{-1}(P)$ and such that the transition monoid of A belongs to $\mathbf{V}$. When the monoid, the morphism and the accepting set are given as inputs, this automaton is log-space computable.*

Proof. It suffices to perform the standard conversion of monoids to finite automata. The set of states of A is M, the initial state is the identity element 1, the transitions are defined by $\delta(m, a) = mh(a)$ for all $m \in M$ and $a \in A$ and the accepting states are P. A straightforward verification shows that the transition monoid of A is isomorphic to M. Since computing images $h(a)$ and performing multiplications are just table lookups, each output bit can be computed in logarithmic time on a random-access machine model. ◀

▶ **Corollary 5.** *For each variety of finite monoids $\mathbf{V}$, the problem* MonIsect($\mathbf{V}$) *is log-space reducible to* DfaIsect($\mathbf{V}$).

For a direct link to Memb($\mathbf{V}$), we consider the variant MonIsect$_1$($\mathbf{V}$) of the finite monoid intersection problem. In this variant, each of the accepting sets is a singleton.

▶ **Proposition 6.** *Let $\mathbf{V}$ be a variety of finite monoids and let $M_1, \ldots, M_k \in \mathbf{V}$ be pairwise disjoint finite monoids. For each $i \in \{1, \ldots, k\}$, let $h_i : A^* \to M_i$ be a morphism and let $p_i \in M_i$. Then there exists a transformation monoid $T \in \mathbf{V}$ on the set $M = M_1 \cup \cdots \cup M_k$, a morphism $h\colon A^* \to T$ and a transformation $p \in T$ such that $h^{-1}(p) = h_1^{-1}(p_1) \cap \cdots \cap h_k^{-1}(p_k)$.*

Proof. For each $a \in A$, we define a transformation $f_a : M \to M$ by $f_a(m) = mh_i(a)$ for $m \in M_i$. The closure of $\{f_a \mid a \in A\}$ under composition is the transformation monoid T and the morphism $h\colon A^* \to T$ is given by $h(a) = f_a$. We let $p\colon M \to M$ be the transformation defined by $p(m) = mp_i$ for $m \in M_i$.

We need to verify that $h^{-1}(p) = h_1^{-1}(p_1) \cap \cdots \cap h_k^{-1}(p_k)$. For the inclusion from right to left, let $w \in A^*$ be a word such that $h_i(w) = p_i$ for each $i \in \{1, \ldots, k\}$. Then, by definition, $h(w)$ is the transformation which maps an element $m \in M_i$ to $mh_i(w) = mp_i$, i.e., $h(w) = p$. The converse inclusion is trivial.

It is easy to check that T is a divisor of the direct product $M_1 \times \cdots \times M_k$ and thus, by closure of $\mathbf{V}$ under direct products and under division, T belongs to $\mathbf{V}$ as well. Since computing images $h_i(a)$ and performing multiplications are just table lookups, each output bit can be computed in logarithmic time on a random-access machine model. ◀

▶ **Corollary 7.** *For each variety of finite monoids* $\mathbf{V}$*, the problem* $\textsc{MonIsect}_1(\mathbf{V})$ *is log-space reducible to* $\textsc{Memb}(\mathbf{V})$*.*

4 Hardness Results

The following lower bound can be viewed as a variant of classical NP-hardness results and is based on the well-known fact that each non-trivial variety contains either the monoid $U_1 = \{0, 1\}$ with integer multiplication or a finite cyclic group (however, the proof itself does not require this case distinction).

▶ **Theorem 8.** *Let* $\mathbf{V}$ *be a non-trivial variety of finite monoids. Then, the decision problem* $\textsc{MonIsect}(\mathbf{V})$ *is* NP*-hard.*

Proof. We give a polynomial-time reduction of the square tiling problem to $\textsc{MonIsect}(\mathbf{V})$.

Let $\mathcal{T} = (\Lambda, T, n, f, b)$ be a tiling system. Let $M \in \mathbf{V}$ be a non-trivial finite monoid and let $x \in M \setminus \{1\}$. The alphabet A is the set $T \times \{1, \ldots, n\} \times \{1, \ldots, n\}$. Let $f = t_1 \cdots t_n$. For each integer $j \in \{1, \ldots, n\}$ and each direction $d \in \{w, e, s, n\}$, we define a morphism $f_{j,d}\colon A \to M$ by mapping $(t, 1, j)$ to x if $\lambda_d(t) = \lambda_d(t_j)$ and mapping the remaining letters to 1. Analogously, with $b = u_1 \cdots u_n$, we let $b_{j,d}\colon A \to M$ be the morphism mapping (t, n, j) to x if $\lambda_d(t) = \lambda_d(u_j)$ and mapping other letters to 1. For each integer $i \in \{1, \ldots, n\}$, each $j \in \{1, \ldots, n-1\}$ and each label $\mu \in \Lambda$, we define a morphism $h_{i,j,\mu}\colon A \to M \times M$ by

$$h_{i,j,\mu}(t,k,\ell) = \begin{cases} (x,1) & \text{if } k = i,\ \ell = j \text{ and } \lambda_e(t) = \mu \\ (1,x) & \text{if } k = i,\ \ell = j+1 \text{ and } \lambda_w(t) = \mu \\ (1,1) & \text{otherwise} \end{cases}$$

and, analogously, we define morphisms $v_{i,j,\mu}\colon A \to M \times M$ with $i \in \{1, \ldots, n-1\}$ and $j \in \{1, \ldots, n\}$ and $\mu \in \Lambda$ as follows:

$$v_{i,j,\mu}(t,k,\ell) = \begin{cases} (x,1) & \text{if } k = i,\ \ell = j \text{ and } \lambda_s(t) = \mu \\ (1,x) & \text{if } k = i+1,\ \ell = j \text{ and } \lambda_n(t) = \mu \\ (1,1) & \text{otherwise} \end{cases}$$

Finally, we define morphisms $g_{i,j,d,\mu,\mu'}\colon A \to M \times M$ with $i, j \in \{1, \ldots, n\}$, $d \in \{w, e, s, n\}$ as well as $\mu, \mu' \in \Lambda$ and $\mu \neq \mu'$ as follows:

$$g_{i,j,d,\mu,\mu'}(t,k,\ell) = \begin{cases} (x,1) & \text{if } k = i,\ \ell = j \text{ and } \lambda_d(t) = \mu \\ (1,x) & \text{if } k = i,\ \ell = j \text{ and } \lambda_d(t) = \mu' \\ (1,1) & \text{otherwise} \end{cases}$$

For each of the morphisms $b_{j,d}$ and $f_{j,d}$, the accepting set is $\{x\}$. For each $h_{i,j,\mu}$ and $v_{i,j,\mu}$, the accepting set is $\{(1,1),(x,x)\}$. The accepting set for each $g_{i,j,d,\mu,\mu'}$ is $\{(1,1),(1,x),(x,1)\}$. ◀

The next objective is to obtain a stronger result in the case that $\mathbf{V}$ contains some finite monoid which is not in $\mathbf{DS}$. Our proof is based on the well-known fact that direct products of B_2^1 can be used to encode computations of a Turing machine or runs of an automaton, an idea which already appears in the proof of [7, Theorem 4.9]. To this end, we first describe classes of languages recognizable by such direct products.

▶ **Lemma 9.** *Let $\mathbf{V}$ be a variety of finite monoids such that $\mathbf{V} \not\subseteq \mathbf{DS}$. Let A be a finite alphabet and let B, C, D, E, F be (possibly empty) pairwise disjoint subsets of A. Then, each of the languages $E^*B(D \cup E)^*$, $(D \cup E)^*CE^*$ and $(E^*B(E \cup F)^*CE^* \cup E^*DE^*)^+$ is the preimage of an element of a monoid $M \in \mathbf{V}$ of size 6 under a morphism $h\colon A^* \to M$.*

Proof. Let N be a monoid from $\mathbf{V} \setminus \mathbf{DS}$. By Proposition 2, the monoid B_2^1 is a divisor of the direct product $N \times N$ and since $\mathbf{V}$ is closed under direct products and divisors, we have $B_2^1 \in \mathbf{V}$. We let $M = B_2^1$.

For $E^*B(D \cup E)^*$, consider the morphism $h\colon A^* \to M$ defined by $h(e) = 1$ for $e \in E$, $h(b) = b$ for $b \in B$, $h(d) = ab$ for all $d \in D$. All other letters are mapped to the zero element. By construction, we have $h^{-1}(b) = E^*B(D \cup E)^*$. For $(D \cup E)^*CE^*$, one can use a symmetrical construction.

For $(E^*B(E \cup F)^*CE^* \cup E^*DE^*)^+$, we define $h\colon A^* \to M$ by $h(b) = a$ for all $b \in B$, $h(c) = b$ for $c \in C$, $h(d) = ab$ for $d \in D$, $h(f) = ba$ for $f \in F$ and $h(e) = 1$ for $e \in E$. Again, the remaining letters are mapped to 0. The preimage of ab is the desired language. ◀

▶ **Lemma 10.** *Let $\mathbf{V}$ be a variety of finite monoids such that $\mathbf{V} \not\subseteq \mathbf{DS}$. Let A be a finite alphabet, let $n \in \mathbb{N}$ and let $A_1, \ldots, A_n$ be pairwise disjoint subsets of A. Then the language $(A_1 \cdots A_n)^+$ can be written as an intersection of n languages, each of which is the preimage of an element of a monoid $M \in \mathbf{V}$ of size 6 under a morphism $h\colon A^* \to M$.*

Proof. Let $B = A_1 \cup \cdots \cup A_n$. For $1 \leqslant i \leqslant n-1$, we define the alphabet $D_i = B \setminus (A_i \cup A_{i+1})$ and the language $L_i = (A_iA_{i+1} \cup D_i)^+$. We also let $L_n = (A_1D_n^*A_n)^+$ with $D_n = B \setminus (A_1 \cup A_n)$. By construction, we have $L_1 \cap \cdots \cap L_n = (A_1 \cdots A_n)^+$ and by Lemma 9, each of the languages L_i is recognized by a monoid of size 6. ◀

We are now able to state the second main theorem of this section.

▶ **Theorem 11.** *Let $\mathbf{V}$ be a variety of finite monoids such that $\mathbf{V} \not\subseteq \mathbf{DS}$. Then, the decision problem* $\textsc{MonIsect}_1(\mathbf{V})$ *is* PSPACE*-complete.*

Proof. Let $\mathcal{T} = (\Lambda, T, n, f, b)$ be a tiling system. The objective is to construct a language L which is non-empty if and only if there exists a valid m-tiling of $\mathcal{T}$ for some $m \in \mathbb{N}$.

We may assume without loss of generality that $\lambda_w(t) \neq \lambda_e(t)$ and $\lambda_s(t) \neq \lambda_n(t)$ for all tiles $t \in T$. If, for example, $\lambda_w(t) = \mu = \lambda_e(t)$ for a tile $t \in T$, we create a copy μ' of the label μ and replace every tile with $\lambda_w(t) = \mu$ by two copies. In one of the copies, we replace the west label with μ'. We repeat this for all other directions and finally remove all tiles with $\lambda_w(t) = \lambda_e(t) \in \{\mu, \mu'\}$.

We define an alphabet $A = T \times \{0,1,2\} \times \{1, \ldots, n\}$. Intuitively, the letters of A correspond to positions in a tiling. The first component describes the tile itself, the second component specifies whether the tile is in the first row, some intermediate row or in the

bottom row and the third component specifies the column. For each $j \in \{1, \ldots, n\}$ and $\mu \in \Lambda$, let $C_j = T \times \{0,1,2\} \times \{j\}$ and $D_j = A \setminus C_j$ and

$$W_\mu = \{(t,i,j) \in A \mid \lambda_w(t) = \mu, j > 1\}, \qquad N_{j,\mu} = \{(t,i,j) \in A \mid \lambda_n(t) = \mu, i > 0\},$$
$$E_\mu = \{(t,i,j) \in A \mid \lambda_e(t) = \mu, j < n\}, \qquad S_{j,\mu} = \{(t,i,j) \in A \mid \lambda_s(t) = \mu, i < 2\},$$
$$X_\mu = A \setminus (W_\mu \cup E_\mu), \qquad Y_{j,\mu} = C_j \setminus (N_{j,\mu} \cup S_{j,\mu}).$$

Note that by our initial assumption, $W_\mu \cap E_\mu = \emptyset$ and $N_{j,\mu} \cap S_{j,\mu} = \emptyset$ for each $\mu \in \Lambda$ and for $1 \leqslant j \leqslant n$. Let $F_j = \{(t_j, 0, j)\}$ and $B_j = \{(u_j, 2, j)\}$ where t_j and u_j are the tiles uniquely determined by $f = t_1 \cdots t_n$ and $b = u_1 \cdots u_n$. Let $\overline{F}_j = \{(t,i,j) \in A \mid i > 0\}$ and $\overline{B}_j = \{(t,i,j) \in A \mid i < 2\}$. We define

$$K = \left(\bigcap_{1 \leqslant j \leqslant n} D_j^* F_j (\overline{F}_j \cup D_j)^* \right) \cap \left(\bigcap_{1 \leqslant j \leqslant n} (\overline{B}_j \cup D_j)^* B_j D_j^* \right) \cap \left(\bigcap_{\mu \in \Lambda} (E_\mu W_\mu \cup X_\mu)^+ \right)$$
$$\cap \left(\bigcap_{\substack{\mu \in \Lambda, \\ 1 \leqslant j \leqslant n}} (D_j^* S_{j,\mu} D_j^* N_{j,\mu} D_j^* \cup D_j^* Y_{j,\mu} D_j^*)^+ \right).$$

and $L = (C_1 \cdots C_n)^+ \cap K$. By Lemma 9 and Lemma 10, the language L can be represented by a MonIsect($\mathbf{V}$) instance with polynomially many morphisms to monoids of size 6 from $\mathbf{V}$ and with singleton accepting sets. ◀

5 A Small Model Property for DO

The objective of this section is to prove the following result which states that, within a non-empty intersection of languages recognized by monoids from **DO**, there always exists a word with a small SLP representation.

▶ **Theorem 12.** *For each $i \in \{1, \ldots, k\}$, let $M_i \in \mathbf{DO}$ and let $h_i \colon A^* \to M_i$ be a morphism. Let $w \in A^*$. Then there exists an SLP S of size at most $p(N)$ with $h_i(S) = h_i(w)$ for all $i \in \{1, \ldots, k\}$ where $p \colon \mathbb{R} \to \mathbb{R}$ is some polynomial and $N = |M_1| + \cdots + |M_k|$.*

Before diving into the proof of this result, we note that the theorem immediately yields the following corollary:

▶ **Corollary 13.** MonIsect(**DO**) *is* NP*-complete.*

Proof. In view of Theorem 8, it suffices to describe an NP-algorithm. The algorithm first non-deterministically guesses an SLP of polynomial size producing a word in the intersection of the given languages. It remains to check that the word represented in the SLP is indeed contained in each of the languages. To this end, we compute the image of the word represented by the SLP under each of the morphisms. Each such computation can be performed in time linear in the size of the SLP by computing the image of a variable X as soon as the images of all variables appearing on the right-hand side of X are computed already, starting with minimal variables. ◀

5.1 The Group Case

We first take care of a special case, namely that each of the monoids is a group. In this case, one can use a variant of the *Schreier-Sims algorithm* [3, 13] to obtain a compressed

representative. To keep the paper self-contained, we give the full algorithm alongside with a correctness proof.

Our setting is as follows: the input are groups $G_1, \ldots, G_k$ which are, without loss of generality, assumed to be pairwise disjoint, and morphisms $h_i \colon A^* \to G_i$ with $i \in \{1, \ldots, k\}$. We let $G = G_1 \cup \cdots \cup G_k$ and $N = |G|$. Note that G is considered as a set; it does not form a group unless $k = 1$. However, for each $g \in G$, we interpret powers g^i in the corresponding group G_i with $g \in G_i$. We let $\omega = N!$ so that, for each $g \in G$, the element g^ω is the identity.[1]

Algorithm 1 The SIFT procedure.

```
procedure SIFT(α)
    R_0 ← ε
    for i ∈ {1, ..., k} do
        S_i ← R_{i-1}^{ω-1} α
        if T[h_i(S_i)] = ε then T[h_i(S_i)] ← S_i end if
        R_i ← R_{i-1} T[h_i(S_i)]
    end for
    return R_k
end procedure
```

The algorithm maintains a table $T \colon G \to (A \cup V)^*$ as an internal data structure, where the set of variables V is extended as needed and the table entries $T[g]$ can be considered variables themselves. The SIFT procedure expects a parameter $\alpha \in (V \cup A)^*$ and tries to find a short representation of $\mathsf{val}(\alpha)$, using only entries from the table unless it comes across an empty table entry, in which case it uses α to fill the missing table entry itself. When a table entry is assigned a word with a factor of the form $X^{\omega-1}$, this factor is stored in a compressed form by using the technique from Lemma 3 and adding new variables as needed. Thus, a factor $X^{\omega-1}$ only requires $4\log(\omega - 1) \leqslant 4\log(N!) \leqslant 4N\log(N)$ additional space.

Algorithm 2 Initialization of the compression algorithm for groups.

```
procedure INIT
    for all g ∈ G do T[g] ← ε end for
    c ← 0
    repeat
        c_p ← c
        for all g_1 ∈ G_1, ... g_k ∈ G_k, a ∈ A do
            SIFT(T[g_1] ··· T[g_k] a)
        end for
        c ← |{g ∈ G | T[g] ≠ ε}|
    until c = c_p
end procedure
```

Before the SIFT procedure is used for compression, the table needs to be initialized. To this end, the INIT routine fills the table with short representatives such that future SIFT invocations never run into empty table entries again. Let us first prove several invariants of the SIFT procedure.

[1] One could also choose $\omega = \operatorname{lcm}\{|G_1|, \ldots, |G_k|\}$ but for the analysis, it does not matter, since $N!$ is sufficiently small.

▶ **Lemma 14.** *For each $i \in \{1, \ldots, k\}$ and $g \in G_i$, we have $T[g] = \varepsilon$ or $h_i(T[g]) = g$.*

Proof. Suppose that $T[g] \neq \varepsilon$. Then, in some round of the SIFT procedure, we have $h_i(S_i) = g$ and $T[h_i(S_i)]$ is assigned the SLP S_i (and never modified again). Therefore, $h_i(T[g]) = h_i(T[h_i(S_i)]) = h_i(S_i) = g$. ◀

▶ **Lemma 15.** *After round i of the* SIFT *procedure, we have $h_i(R_i) = h_i(\alpha)$.*

Proof. By the definition of R_i, we have $h_i(R_i) = h_i(R_{i-1}T[h_i(S_i)])$ which is the same as $h_i(R_{i-1}S_i)$ by Lemma 14. Plugging in the definition of S_i yields $h_i(R_{i-1}R_{i-1}^{\omega-1}\alpha) = h_i(\alpha)$ where the latter equality holds since G_i is a group. ◀

▶ **Lemma 16.** *For $1 \leqslant i < j \leqslant k$ and for all $g \in G_j$, we have $h_i(T[g]) = 1$.*

Proof. Consider the invocation of the SIFT procedure where $T[g]$ is defined. In round j of this invocation, the entry $T[g]$ is assigned some SLP S_j with $h_j(S_j) = g$.

Therefore, $h_i(T[g]) = h_i(S_j) = h_i(R_{j-1}^{\omega-1}\alpha)$. Expanding R_{j-1} yields

$$h_i(R_{j-1}) = h_i\Big(\Big(\prod_{r=1}^{j-1} T[h_r(S_r)]\Big)\Big) = h_i\Big(\Big(\prod_{r=1}^{i} T[h_r(S_r)]\Big)\Big) = h_i(R_i)$$

where the second equality follows by induction. Therefore, $h_i(T[g]) = h_i(R_i^{\omega-1}\alpha)$ which is the same as $h_i(\alpha^{\omega-1}\alpha) = 1$ by Lemma 15. ◀

▶ **Lemma 17.** *After round j, we have $h_i(R_k) = h_i(\alpha)$ for all $i \in \{1, \ldots, j\}$.*

Proof. Using the expansion of R_k and Lemma 16, we obtain the sequence of equalities

$$h_i(R_k) = h_i\Big(\Big(\prod_{r=1}^{k} T[h_r(S_r)]\Big)\Big) = h_i\Big(\Big(\prod_{r=1}^{i} T[h_r(S_r)]\Big)\Big) = h_i(R_i).$$

The statement now follows immediately from Lemma 15. ◀

▶ **Theorem 18.** *For each $i \in \{1, \ldots, k\}$, let G_i be a finite group and let $h_i \colon A^* \to G_i$ be a morphism. Let $w \in A^*$. Then there exists an SLP S of size at most $p(N)$ with $h_i(S) = h_i(w)$ for all $i \in \{1, \ldots, k\}$ where $p \colon \mathbb{R} \to \mathbb{R}$ is some polynomial and $N = |G_1| + \cdots + |G_k|$.*

Proof. We claim that the SLP S constructed when calling INIT, followed by SIFT with parameter w satisfies the properties above. By Lemma 17, we have $h_i(S) = h_i(w)$ for all $i \in \{1, \ldots, k\}$. Moreover, when the initialization routine returns, the table entries contain SLP of polynomially bounded size. We now claim that any subsequent executions of the SIFT procedure will not define any new table entries, no matter which SLP is passed as a parameter. In particular, running SIFT(w) yields an SLP that only uses already existing table entries.

To prove the claim, assume, for the sake of contradiction, that there exists some word v such that some new table entry $T[g]$ is defined during SIFT(v). We choose v such that it is a word of minimal length satisfying this condition. This means that we can factorize $v = v'a$ with $a \in A$ such that all table entries are defined when calling SIFT(v'). Let $T[g_1] \cdots T[g_k]$ be the return value of SIFT(v'). Then SIFT$(T[g_1] \cdots T[g_k]a)$ is called during the initialization process and because $h_i(T[g_1] \cdots T[g_k]a) = h_i(v)$ for all $1 \leqslant i \leqslant k$, the sequence of S_i during the execution of SIFT$(T[g_1] \cdots T[g_k]a)$ is the same as in SIFT(v) which means that all table entries accessed during SIFT(v) are defined. ◀

5.2 The General Case

For the general case, where each of the monoids is in **DO** but not necessarily a group, we use combinatorial properties of languages recognized by monoids from **DO** to reduce the problem to the group case. The following lemmas are an essential ingredient of this reduction.

▶ **Lemma 19.** *Let $h\colon A^* \to M$ be a morphism to a finite monoid $M \in \mathbf{DS}$. Let $u, v \in A^*$ such that $h(v) \in E(M)$ and $\mathsf{alph}(u) \subseteq \mathsf{alph}(v)$. Then $h(v) \leqslant_{\mathcal{J}} h(u)$.*

Proof. Let $u = a_1 \cdots a_\ell$ with $a_i \in A$ for $1 \leqslant i \leqslant \ell$. Since $a_i \in \mathsf{alph}(v)$ for each $i \in \{1, \ldots, \ell\}$, we have $h(v) \leqslant_{\mathcal{J}} h(a_i)$. By Proposition 2, the set $\{x \in M \mid h(v) \leqslant_{\mathcal{J}} x\}$ is a submonoid of M which means that $h(v) \leqslant_{\mathcal{J}} h(a_1) \cdots h(a_\ell) = h(u)$, thereby proving the claim. ◀

▶ **Lemma 20.** *Let $M \in \mathbf{DO}$ and let $e, f, g \in E(M)$ with $e \mathcal{J} f \leqslant_{\mathcal{J}} g$. Then $egf = ef$.*

Proof. First note that $(fg)^\omega \mathcal{J} (fgf)^\omega \mathcal{J} (gf)^\omega$. Since $M \in \mathbf{DS}$, we have $(fgf)^\omega = f$ and since $M \in \mathbf{DO}$, we have $(fg)^\omega = (fg)^\omega (gf)^\omega (fg)^\omega = (fg)^{\omega-1}$. Together, this yields, $fgf = (fgf)^\omega gf = (fg)^\omega fgf = (fg)^{\omega-1} fgf = (fg)^\omega f = (fgf)^\omega = f$, thus $gf \in E(M)$. By Proposition 2, we obtain $gf \mathcal{J} e$. Therefore, $egf = eg(fef) = (egfe)f = ef$. ◀

▶ **Lemma 21.** *Let $M \in \mathbf{DO}$, let $e, f, g \in E(M)$ and let $x, y \in M$ such that $e \mathcal{J} f \leqslant_{\mathcal{J}} g, x, y$. Then $exgyf = exyf$.*

Proof. Since $M \in \mathbf{DS}$, we have $ex = (exe)^\omega x = ex(ex)^\omega$ and $yf = y(fyf)^\omega = (yf)^\omega yf$. Note that $(ex)^\omega \mathcal{J} e \mathcal{J} f \mathcal{J} (yf)^\omega$ by Proposition 2 and thus, Lemma 20 yields $(ex)^\omega g (yf)^\omega = (ex)^\omega (yf)^\omega$. Finally, combining all the equalities, we obtain the desired statement $exgyf = ex(ex)^\omega g (yf)^\omega yf = ex(ex)^\omega (yf)^\omega yf = exyf$. ◀

For the remainder of this section, let $M_1, \ldots, M_k \in \mathbf{DO}$ be finite monoids and let $h_i\colon A^* \to M_i$ be morphisms. We let $N = |M_1| + \cdots + |M_k|$. The occurrence of a word u in puq is called *isolated* if for each $i \in \{1, \ldots, k\}$, there exist words $v_i, w_i \in A^*$ such that

$$\mathsf{alph}(v_i) = \mathsf{alph}(w_i) \supseteq \mathsf{alph}(u), \quad h_i(pv_i) = h_i(p) \quad \text{and} \quad h_i(w_i q) = h_i(q).$$

Let $w = a_1 u_1 a_2 \cdots u_{\ell-1} a_\ell$ be a factorization of w with $a_j \in A$ and $u_j \in A^*$ for all $j \in \{1, \ldots, \ell\}$. Let $p_j = a_1 u_1 a_2 \cdots u_{j-1} a_j$ and $q_j = a_{j+1} u_{j+1} \cdots a_{\ell-1} u_{\ell-1} a_\ell$. The factorization $w = a_1 u_1 a_2 \cdots u_{\ell-1} a_\ell$ is called *piecewise isolating* if, for each $j \in \{1, \ldots, \ell - 1\}$, the occurrence of u_j in $w = p_j u_j q_j$ is isolated. The value ℓ is the *length* of this factorization.

▶ **Lemma 22.** *Every word $w \in A^*$ admits a piecewise isolating factorization of length at most N^2.*

Proof. Let $w = b_1 \cdots b_m$ where $b_r \in A$ for $1 \leqslant r \leqslant m$. To each position $r \in \{1, \ldots, m\}$, we assign a set $C_r = \{(h_i(b_1 \cdots b_s), h_i(b_{s+1} \cdots b_m)) \mid 1 \leqslant i \leqslant k, 1 \leqslant s \leqslant r\}$. Note that by definition, we have $C_r \subseteq C_{r+1}$. Let $r_1, \ldots, r_\ell \in \mathbb{N}$ such that $r_1 = 1$, $r_\ell = m$ and $C_{r_{j-1}} = C_{r_j - 1} \subsetneq C_{r_j}$ for all $j \in \{2, \ldots, \ell\}$. Let $a_j = b_{r_j}$ and let $u_j = b_{r_j + 1} \cdots b_{r_{j+1} - 1}$ for all $j \in \{1, \ldots, \ell\}$.

Now, for $j \in \{1, \ldots, \ell\}$ and $i \in \{1, \ldots, k\}$, let $t(j, i)$ be the smallest index g such that $(h_i(a_1 u_1 \cdots a_j u_j), h_i(a_{j+1} u_{j+1} \cdots a_{\ell-1} u_{\ell-1} a_\ell)) \in C_{r_g}$, i.e., the prefix of length $r_{t(j,i)}$ of w is the shortest prefix p such that $w = pq$ for some $q \in A^*$ and the image of p under h_i is $h_i(a_1 u_1 \cdots a_j u_j)$ and the image of q is $h_i(a_{j+1} u_{j+1} \cdots a_{\ell-1} u_{\ell-1} a_\ell)$. Note that $t(j, i) \leqslant j$ and, by choice of $t(j, i)$, we have

$$h_i(a_1 u_1 \cdots a_j u_j) = h_i(a_1 u_1 a_2 \cdots u_{t(j,i)-1} a_{t(j,i)}) \text{ and} \tag{1}$$

$$h_i(a_{j+1} u_{j+1} \cdots a_{\ell-1} u_{\ell-1} a_\ell) = h_i(u_{t(j,i)} a_{t(j,i)+1} \cdots u_{\ell-1} a_\ell). \tag{2}$$

Let $w_{ji} = u_{t(j,i)}a_{t(j,i)+1}\cdots u_{j-1}a_ju_j$ and let $v_{ji} = u_ju_{t(j,i)}a_{t(j,i)+1}\cdots u_{j-1}a_j$. In the special case $t(j,i) = j$, we obtain $w_{ji} = v_{ji} = u_j$.

For $p_j = a_1u_1a_2\cdots u_{j-1}a_j$ and $q_j = a_{j+1}u_{j+1}\cdots a_{\ell-1}u_{\ell-1}a_\ell$, equation (1) implies

$$\begin{aligned} h_i(p_jv_{ji}) &= h_i(a_1u_1\cdots a_ju_j)\cdot h_i(u_{t(j,i)}a_{t(j,i)+1}\cdots u_{j-1}a_j) \\ &= h_i(a_1u_1a_2\cdots u_{t(j,i)-1}a_{t(j,i)})\cdot h_i(u_{t(j,i)}a_{t(j,i)+1}\cdots u_{j-1}a_j) = h_i(p_j) \end{aligned}$$

and, similarly, equation (2) yields $h_i(w_{ji}q_j) = h_i(q_j)$. Since u_j is a suffix of w_{ji} and since v_{ji} can be obtained by rotating w_{ji} cyclically, we have $\mathsf{alph}(v_{ji}) = \mathsf{alph}(w_{ji}) \supseteq \mathsf{alph}(u_j)$. The bound on ℓ follows from the fact that $C_{r_1} \subsetneq \cdots \subsetneq C_{r_\ell} \subseteq \bigcup_{i=1}^{k} M_i \times M_i$. ◀

The lemma above suggests that it is sufficient to construct SLPs for isolated occurrences. Thus, let now $u \in A^*$ be an isolated occurrence of $w = puq$, and let $B = \mathsf{alph}(u)$. For each $i \in \{1,\ldots,k\}$, we define an equivalence relation $\equiv_i$ on the submonoid $T_i = h_i(B^*)$ of M_i by $m \equiv_i n$ if and only if $h_i(p)xmyh_i(q) = h_i(p)xnyh_i(q)$ for all $x, y \in T_i$. It is easy to check that this relation is a congruence. Moreover, for all $u, v \in B^*$ with $h_i(u) \equiv_i h_i(v)$, we have $h_i(puq) = h_i(pvq)$. Another fundamental property of $\equiv_i$ is captured in the following lemma.

▶ **Lemma 23.** *For each $i \in \{1,\ldots,k\}$, the quotient $T_i/{\equiv_i}$ is a group.*

Proof. Let $\omega = N!$ and let $m \in T_i$ be an arbitrary element. It suffices to show that $m^\omega \equiv_i 1$, i.e., for all $x, y \in T_i$, we have $h_i(p)xm^\omega yh_i(q) = h_i(p)xyh_i(q)$.

Let $v_i, w_i \in A^*$ as in the definition of isolated occurrences and let $e = h(v_i^\omega)$ and $f = h(w_i^\omega)$. Note that $h_i(pv_i) = h_i(p)$ implies $h_i(p)e = h_i(p)$. Analogously, we have $fh_i(q) = h_i(q)$. Since B is contained in $\mathsf{alph}(v_i) = \mathsf{alph}(w_i)$ and since $m, x, y \in T_i = h_i(B^*)$, we have $e \mathrel{\mathcal{J}} f \leqslant_{\mathcal{J}} m^\omega, x, y$ by Lemma 19. Therefore,

$$h_i(p)xm^\omega yh_j(q) = h_i(p)exm^\omega yfh_i(q) = h_i(p)exyfh_i(q) = h_i(p)xyh_i(q)$$

where the second equality uses Lemma 21. ◀

We now return to the proof of the main theorem of this section.

Proof of Theorem 12. By considering a piecewise isolating factorization of w, it suffices to show that if u is an isolated occurrence in $w = puq$, then there exists an SLP S of polynomial size with $h_i(pSq) = h_i(puq)$ for all $i \in \{1,\ldots,k\}$. Combining the letters a_i and the SLPs for the isolated occurrences in the piecewise isolating factorization, we obtain the SLP for w.

Let again $B = \mathsf{alph}(u)$. To obtain a polynomial-size SLP S with $h_i(pSq) = h_i(puq)$ for all $i \in \{1,\ldots,k\}$, we consider the morphisms $\psi_i\colon B^* \to T_i/{\equiv_i}$ defined by $\psi_i(v) = [h_i(v)]_{\equiv_i}$, i.e., each word v is mapped to the equivalence class of $h_i(v)$ with respect to $\equiv_i$. Note that $|T_i/{\equiv_i}| \leqslant |T_i| \leqslant |M_i|$ for $1 \leqslant i \leqslant k$ and by Lemma 23, each of the monoids $T_i/{\equiv_i}$ is a group. By Theorem 18, there exists a polynomial-size SLP S with $\psi_i(S) = \psi_i(u)$ for all $i \in \{1,\ldots,k\}$ and, by the definition of $\equiv_i$, we obtain $h_i(pSq) = h_i(puq)$, as desired. ◀

6 Summary and Outlook

We investigated the complexity of the intersection problem for finite monoids, showing that the problem is NP-complete for varieties contained in **DO** and PSPACE-complete for varieties not contained within **DS**. To obtain a dichotomy result, one needs to investigate the complexity of the problem when monoids from $\mathbf{DS} \setminus \mathbf{DO}$ are part of the input. Using techniques similar to those in Section 5, we were able to show that for a subset of this class,

the problem remains NP-complete and thus, we conjecture that the problem is NP-complete whenever $\mathbf{V} \subseteq \mathbf{DS}$. The fact that $\mathbf{DS} \setminus \mathbf{DO}$ have not been studied and understood well enough from a language-theoretic perspective makes the problem of classifying the complexity of these monoids challenging but, at the same time, an interesting object for further research.

References

1. Jorge Almeida. *Finite Semigroups and Universal Algebra*. World Scientific, Singapore, 1994.
2. László Babai, Eugene M. Luks, and Ákos Seress. Permutation groups in NC. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 409–420. ACM, 1987. doi:10.1145/28395.28439.
3. László Babai, Eugene M. Luks, and Ákos Seress. Permutation groups in NC. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 409–420. ACM, 1987. doi:10.1145/28395.28439.
4. Martin Beaudry. Membership testing in commutative transformation semigroups. *Inf. Comput.*, 79(1):84–93, 1988. doi:10.1016/0890-5401(88)90018-1.
5. Martin Beaudry. *Membership Testing in Transformation Monoids*. PhD thesis, McGill University, Montreal, Quebec, 1988.
6. Martin Beaudry. Membership testing in threshold one transformation monoids. *Inf. Comput.*, 113(1):1–25, 1994. doi:10.1006/inco.1994.1062.
7. Martin Beaudry, Pierre McKenzie, and Denis Thérien. The membership problem in aperiodic transformation monoids. *J. ACM*, 39(3):599–616, 1992. doi:10.1145/146637.146661.
8. László Bernátsky. Regular expression star-freeness is PSPACE-complete. *Acta Cybernetica*, 13(1):1–21, 1997.
9. Michael Blondin, Andreas Krebs, and Pierre McKenzie. The complexity of intersecting finite automata having few final states. *Computational Complexity*, 25(4):775–814, 2016. doi:10.1007/s00037-014-0089-9.
10. Bogdan S. Chlebus. Domino-tiling games. *J. Comput. Syst. Sci.*, 32(3):374–392, 1986. doi:10.1016/0022-0000(86)90036-X.
11. Sung Cho and Dung T. Huynh. Finite automaton aperiodicity is PSPACE-complete. *Theoretical Computer Science*, 88:96–116, 1991.
12. Volker Diekert, Claudio Gutiérrez, and Christian Hagenah. The existential theory of equations with rational constraints in free groups is PSPACE-complete. *Information and Computation*, 202:105–140, 2005. Conference version in STACS 2001, LNCS 2010, 170–182, 2004.
13. Merrick L. Furst, John E. Hopcroft, and Eugene M. Luks. Polynomial-time algorithms for permutation groups. In *21st Annual Symposium on Foundations of Computer Science, Syracuse, New York, USA, 13-15 October 1980*, pages 36–41. IEEE Computer Society, 1980. doi:10.1109/SFCS.1980.34.
14. Markus Holzer and Martin Kutrib. Descriptional and computational complexity of finite automata - A survey. *Inf. Comput.*, 209(3):456–470, 2011. doi:10.1016/j.ic.2010.11.013.
15. Tao Jiang and Bala Ravikumar. Minimal NFA problems are hard. In Javier Leach Albert, Burkhard Monien, and Mario Rodríguez-Artalejo, editors, *ICALP*, volume 510 of *Lecture Notes in Computer Science*, pages 629–640. Springer, 1991.
16. Dexter Kozen. Lower bounds for natural proof systems. In *Proc. of the 18th Ann. Symp. on Foundations of Computer Science, FOCS'77*, pages 254–266, Providence, Rhode Island, 1977. IEEE Computer Society Press.

17 Klaus-Jörn Lange and Peter Rossmanith. The emptiness problem for intersections of regular languages. In Ivan M. Havel and Václav Koubek, editors, *Mathematical Foundations of Computer Science 1992, 17th International Symposium, MFCS'92, Prague, Czechoslovakia, August 24-28, 1992, Proceedings*, volume 629 of *Lecture Notes in Computer Science*, pages 346–354. Springer, 1992. doi:10.1007/3-540-55808-X_33.

18 Jean-Éric Pin. *Varieties of Formal Languages*. North Oxford Academic, London, 1986.

19 Marcel-Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8:190–194, 1965.

20 Pascal Tesson and Denis Thérien. Diamonds are forever: The variety DA. In Gracinda Maria dos Gomes Moreira da Cunha, Pedro Ventura Alves da Silva, and Jean-Éric Pin, editors, *Semigroups, Algorithms, Automata and Languages, Coimbra (Portugal) 2001*, pages 475–500. World Scientific, 2002.