Efficient Algorithms for Morphisms over Omega-Regular Languages

Lukas Fleischer and Manfred Kufleitner

FMI, University of Stuttgart*
Universitätsstraße 38, 70569 Stuttgart, Germany
{fleischer,kufleitner}@fmi.uni-stuttgart.de

Abstract

Morphisms to finite semigroups can be used for recognizing omega-regular languages. The so-called strongly recognizing morphisms can be seen as a deterministic computation model which provides minimal objects (known as the syntactic morphism) and a trivial complementation procedure. We give a quadratic-time algorithm for computing the syntactic morphism from any given strongly recognizing morphism, thereby showing that minimization is easy as well. In addition, we give algorithms for efficiently solving various decision problems for weakly recognizing morphisms. Weakly recognizing morphism are often smaller than their strongly recognizing counterparts. Finally, we describe the language operations needed for converting formulas in monadic second-order logic (MSO) into strongly recognizing morphisms, and we give some experimental results.

1998 ACM Subject Classification F.4.1 Mathematical Logic; F.4.3 Formal Languages

Keywords and phrases Büchi automata, omega-regular language, syntactic semigroup, recognizing morphism, MSO

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

Automata over finite words have a huge number of effective closure properties. Moreover, many problems such as minimization or equivalence of deterministic automata admit very efficient algorithms [6, 7]. The situation over infinite words is quite similar, but with the major difference that many operations are less efficient. There are many different automaton models for accepting languages of infinite words, the so-called ω-regular languages. Each of these models has its advantages and disadvantages. For instance, deterministic Büchi automata are less powerful than nondeterministic Büchi automata [15]. And only very few automaton models admit efficient minimization algorithms; for example, minimization of deterministic finite automata can be applied to the lasso automata in [2].

The theory of finite semigroups and automata is tightly connected [11]. Since the semigroup for a language can be exponentially bigger than its automaton, semigroups have very rarely been considered in the context of efficient algorithms. There is also an algebraic approach to ω-regular languages by using morphisms to finite semigroups, see e.g. [9, 15]. Among the many nice properties of this approach are minimal morphisms — the so-called syntactic morphisms — and easy complementation. As for finite words, the semigroup for an ω-regular language can be exponentially bigger than its Büchi automaton. However, since many operations for ω-regular languages are less efficient than for regular languages over

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

Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 1–13

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

finite words, the drawback of this exponential blow-up in size is less serious. This is even more so when minimizing all intermediate objects.

A typical algorithm for computing the syntactic morphism of a regular language over finite words is to minimize the (deterministic) automaton defined by the Cayley graph of a morphism, and then the syntactic morphism is given by the transition semigroup of the minimal automaton. This approach does not work for infinite words and we therefore give a direct algorithm for computing the syntactic morphism. Our algorithm is an adaptation of Hopcroft's minimization algorithm [6] and its running time is quadratic in the size of the semigroup. We show that this is rather optimal.

There are two different modes for recognizing omega-regular languages by a morphism to a finite semigroup: *weak* and *strong* recognition. Strong recognition is a special case of weak recognition. Easy complementation and the computation of the syntactic morphism only works for strong recognition. We show how to test whether a given weak recognition is actually strong. Another useful tool for morphisms is the computation of the so-called conjugacy classes.

As an application, we consider the translation of MSO formulas into strongly recognizing morphisms. To this end, we show that a powerset construction preserves strong recognition, and that this construction can be used for computing the image under a length-preserving morphism. Finally, we give the test results of some translations from MSO to strong recognition. Deciding the satisfiability of an MSO formula is non-elementary [13] and therefore, minimization of intermediate objects is usually very helpful for solving some special cases. This is confirmed by our test results.

2 Preliminaries

Words. Let A be a finite *alphabet*. The elements of A are called *letters*. A *finite word* is a sequence $a_1a_2\cdots a_n$ of letters of A and an *infinite word* is an infinite sequence $a_1a_2\cdots$. The empty word is denoted by ε. The set of non-empty finite words over A is A^+. Let K be a set of finite words and let L be a set of infinite words. We set $KL = \{u\alpha \mid u \in K, \alpha \in L\}$, $K^+ = \{u_1u_2\cdots u_n \mid n \geqslant 1, u_i \in K\}$ and $K^* = K^+ \cup \{\varepsilon\}$. Moreover, if $\varepsilon \notin K$ we define the *infinite iteration* $K^\omega = \{u_1u_2\cdots \mid u_i \in K\}$. A natural extension to $K \subseteq A^*$ is $K^\omega = (K \setminus \{\varepsilon\})^\omega$.

Finite semigroups. Let S be a finite semigroup. An element e of S is *idempotent* if $e^2 = e$. The set of idempotent elements of S is denoted by $E(S) = \{e \in S \mid e^2 = e\}$. For each $s \in S$ the set $\{s^k \mid k \geqslant 1\}$ of all powers of s is finite and it contains exactly one idempotent element.

A semigroup S is called *X-generated* if X is a subset of S and every element of S can be written as a product of elements of X. The *right Cayley graph* of an X-generated semigroup S has S as vertices and its labeled edges are the triples of the form (s, a, sa) for $s \in S$ and $a \in X$. The *left Cayley graph* of S is defined analogously with edges of the form (s, a, as). The definitions of Cayley graphs depend on the choice of the set X. In the following, when a surjective morphism $h\colon A^+ \to S$ is given, we choose $X = h(A)$ as the set of generators.

Green's relations are an important tool in the study of finite semigroups. We denote by S^1 the monoid that is obtained by adding a new neutral element 1 to S. For $s, t \in S$ let

$$s \mathcal{R} t \text{ if there exist } q, q' \in S^1 \text{ such that } sq = t \text{ and } tq' = s,$$
$$s \mathcal{L} t \text{ if there exist } p, p' \in S^1 \text{ such that } ps = t \text{ and } p't = s.$$

These relations are equivalence relations and the equivalence classes of $\mathcal{R}$ (resp. $\mathcal{L}$) are called *$\mathcal{R}$-classes* (resp. *$\mathcal{L}$-classes*). The $\mathcal{R}$-classes (resp. $\mathcal{L}$-classes) of a semigroup S can be

computed in time linear in $|S|$ by applying Tarjan's algorithm to the right (resp. left) Cayley graph of S, see [5].

An element $(s, e) \in S \times E(S)$ is a *linked pair* if $se = s$. Two linked pairs (s, e) and (t, f) are *conjugate*, written as $(s, e) \sim (t, f)$, if there exist $x, y \in S$ such that $sx = t$, $xy = e$ and $yx = f$. The conjugacy relation $\sim$ on the set of linked pairs is an equivalence relation, see e.g. [9]. The equivalence classes of $\sim$ are called *conjugacy classes*. A set P of linked pairs is *closed under conjugation* if it is a union of conjugacy classes.

Recognition by morphisms. A language $L \subseteq A^\omega$ is *regular* (or *ω-regular*) if it is recognized by some finite Büchi automaton, see e.g. [3]. The family of regular languages is closed under Boolean operations, i.e., set union, set intersection and complementation. We now describe algebraic recognition modes for regular languages. Let $h\colon A^+ \to S$ be a morphism onto a finite semigroup S. For $s \in S$, we set $[s] = h^{-1}(s)$ and for $P \subseteq S \times S$, we set

$$[P] = \bigcup_{(s,t)\in P} [s][t]^\omega$$

if h is understood from the context. A language $L \subseteq A^\omega$ is *weakly recognized* by a morphism $h : A^+ \to S$ if there exists a set of linked pairs $P \subseteq S \times E(S)$ with $L = [P]$. If in addition P is closed under conjugation, then h *strongly recognizes* L. Another well-known characterisation of strong recognition is the following, see e.g. [4].

▶ **Proposition 1.** *Let $h : A^+ \to S$ be a morphism onto a finite semigroup. Then h strongly recognizes L if and only if $[s][t]^\omega \cap L \neq \emptyset$ implies $[s][t]^\omega \subseteq L$ for all $s, t \in S$.*

The *syntactic congruence* $\equiv_L$ of a language $L \subseteq A^\omega$ is defined over A^+ as $u \equiv_L v$ if the equivalences

$$(xuy)z^\omega \in L \Leftrightarrow (xvy)z^\omega \in L \text{ and}$$
$$z(xuy)^\omega \in L \Leftrightarrow z(xvy)^\omega \in L$$

hold for all finite words $x, y, z \in A^*$. Our definition is slightly different but equivalent to the syntactic congruence introduced by Arnold [1]. The congruence classes of $\equiv_L$ form the so-called *syntactic semigroup* $A^+/\equiv_L$ and the *syntactic morphism* $h_L\colon A^+ \to A^+/\equiv_L$ is the natural quotient map. If L is regular, the syntactic semigroup of L is finite and h_L strongly recognizes L [1, 9].

Model of computation. Morphisms $h\colon A^+ \to S$ are given implicitly through a mapping $f\colon A \to S$ with $f(a) = h(a)$ for all $a \in A$. We assume that for finite semigroups S, multiplications can be performed in constant time. Some algorithms only perform multiplications of the form $h(a) \cdot s$ or $s \cdot h(a)$ where h is a morphism, s is an element of S and a is a letter. In that case, semigroups can be represented efficiently by their left and right Cayley graphs. For two elements $s, t \in S$ we can check in constant time whether $s = t$ and it is possible to organize elements of S in a hash map such that operations on subsets of S can be implemented efficiently. When a set $P \subseteq S \times S$ is part of the input, we assume that for each $s, t \in S$ one can check in constant time whether $(s, t) \in P$.

3 Conversion between Büchi automata, weak and strong recognition

In this section, we describe well-known constructions for the conversion between the different acceptance modes for regular languages. For details and proofs, we refer to [9, 10, 15].

3.1 From Büchi automata to strong recognition

In the case of finite words, when proving that each regular language is recognizable by a morphism onto a finite semigroup, one usually considers the transition semigroup of a finite automaton. However, when applying the same construction to Büchi automata, the resulting morphism only weakly recognizes the language. In this section, we describe a construction to convert a Büchi automaton $\mathcal{A} = (Q, A, \delta, I, F)$ into a semigroup S and a morphism $h\colon A^+ \to S$ that strongly recognizes $L(\mathcal{A})$.

For states $p, q \in Q$ and a finite word $u \in A^+$, we write $p \xrightarrow{u} q$ if there exists a sequence $q_0 a_1 q_1 a_2 q_2 \cdots q_{n-1} a_n q_n$ with $q_0 = p$, $q_n = q$ and $(q_i, a_{i+1}, q_{i+1}) \in \delta$ for all $i \in \{0, \ldots, n-1\}$. If, additionally, $q_i \in F$ for some $i \in \{0, \ldots, n\}$, we write $p \xrightarrow[F]{u} q$. We now assign to each word $u \in A^+$ a $Q \times Q$ matrix $h(u)$ defined by

$$(h(u))_{pq} = \begin{cases} 1 & \text{if } p \xrightarrow{u} q \text{ but not } p \xrightarrow[F]{u} q \\ 2 & \text{if } p \xrightarrow[F]{u} q \\ 0 & \text{otherwise} \end{cases}$$

A routine verification shows that this naturally extends the image of A^+ under h to a semigroup S. We say that a linked pair (R, E) where $R = (r_{pq})_{p,q \in Q}$ and $E = (e_{pq})_{p,q \in Q}$ is *accepting* if there exist states $p, q \in Q$ such that $r_{pq} \geqslant 1$ and $e_{qq} = 2$. One can now verify that the set P of all accepting linked pairs is closed under conjugation and that $[P] = L(\mathcal{A})$.

3.2 From weak recognition to Büchi automata

Suppose we are given a morphism $h\colon A^+ \to S$ onto a finite semigroup S that weakly recognizes a language L, i.e., $L = [P]$ for some set of linked pairs $P \subseteq S \times E(S)$. One can use the following construction from [10] to obtain a Büchi automaton $\mathcal{A}$ with $L(\mathcal{A}) = L$.

The set of states is $Q = S^1 \times E(S)$, the set of initial states is $I = P$ and the set of final states is $F = \{1\} \times E(S)$. The transition relation δ consists of all tuples of the form $((s, e), a, (t, e)) \in Q \times A \times Q$ where $h(a)t = s$ or $h(a)t = se$.

By combining the constructions from this and the previous subsection, we also obtain a construction to convert a morphism that weakly recognizes a language L into a morphism that strongly recognizes L. There are also direct, more efficient constructions, to perform this conversion, see e.g. [9]. The converse direction is trivial since, by definition, a morphism $h\colon A^+ \to S$ that strongly recognizes a language L also weakly recognizes L.

4 Computing conjugacy classes

When designing an algorithm that takes a set of linked pairs $P \subseteq S \times E(S)$ as input, it is often convenient to assume that P is closed under conjugation. However, this is not always the case in practice: The input set P might be a proper subset of its closure under conjugation Q such that $[P] = [Q]$. In this section, we describe an algorithm to compute the conjugacy classes efficiently. It justifies the assumption that P is always closed under conjugation in the following sections, particularly in Section 6.

As a warm-up, we first describe how to compute the set F of linked pairs. The linked pairs are exactly the pairs of the form (se, e) with $s \in S$ and $e \in E(S)$. Thus, we first check for each element $e \in S$ whether $e^2 = e$. If the outcome of the check is positive, we perform a depth-first search in the left Cayley graph of S, starting at element e. For each element s that is visited, (s, e) is a linked pair. The total running time of this routine is $\mathcal{O}(|S| + |A| \cdot |F|)$.

An equivalence relation $\equiv$ on the set of linked pairs is called *left-stable* if for all $p \in S$ and for linked pairs (s, e), (t, f) with $(s, e) \equiv (t, f)$, we have $(ps, e) \equiv (pt, f)$. We define an equivalence relation $\approx$ on the set of linked pairs by $(s, e) \approx (t, f)$ if and only if $e \mathcal{L} s \mathcal{R} t \mathcal{L} f$ or $(s, e) = (t, f)$. Its relationship to conjugacy is captured in the following Lemma:

▶ **Lemma 2.** *The conjugacy relation $\sim$ is the finest left-stable equivalence relation coarser than $\approx$.*

Proof. It follows directly from the definitions of linked pairs and conjugacy that $\sim$ is left-stable. Let (s, e) and (t, f) be linked pairs with $(s, e) \approx (t, f)$ and $(s, e) \neq (t, f)$. Since $s \mathcal{R} t$, there exist $q, q' \in S^1$ such that $sq = t$ and $tq' = s$. We set $x = eq$ and $y = fq'$. Now, $sx = seq = sq = t$. Moreover, since $s \mathcal{L} e$, there exists $p \in S^1$ with $ps = e$. Thus, we have $xy = eqy = psqy = pty = ptfq' = ptq' = ps = e$. A similar argument can be used to show that $yx = f$. Hence, (s, e) and (t, f) are conjugate, and $\sim$ is indeed coarser than $\approx$.

In order to show that $\sim$ is the finest relation with these properties, we consider an arbitrary left-stable equivalence relation $\simeq$ on the set of linked pairs which is coarser than $\approx$. We show that $(s, e) \sim (t, f)$ implies $(s, e) \simeq (t, f)$. Let $x, y \in S$ such that $sx = t$, $xy = e$ and $yx = f$. Then we have $ex = xyx = xf$ and $xfy = xyxy = e^2 = e$, which shows that $e \mathcal{R} xf$. Furthermore we have $xf \mathcal{L} f$, since $yxf = f^2 = f$. By the definition of $\approx$, this means that $(e, e) \approx (xf, f)$ and since $\approx$ refines $\simeq$, it follows that $(e, e) \simeq (xf, f)$. Left-stability yields $(s, e) = (se, e) \simeq (sxf, f) = (t, f)$. ◀

Since $\mathcal{R}$-classes and $\mathcal{L}$-classes can be computed in time linear in the size of the semigroup, this allows us to efficiently compute the conjugacy classes as shown in Algorithm 1. We use a so-called *disjoint-set data structure* that provides two operations on a partition. Find(s, e) returns a unique element from the class that contains (s, e), i.e., if (s, e) and (t, f) are in the same class, we have Find$(s, e) =$ Find(t, f). Union$((s, e), (t, f))$ merges the classes of (s, e) and (t, f). To simplify the notation we also introduce an operation Union$^+(R)$ for subsets R of $S \times S$ that merges all classes with elements in R. Union$^+(R)$ can be implemented using $|R| - 1$ atomic Union operations. The partition is initialized with singleton sets $\{(s, e)\}$ for all linked pairs (s, e). The second data structure used in the algorithm is a set $T \subseteq 2^F$.

Algorithm 1 Computing conjugacy classes

```
initialize T with the non-trivial equivalence classes of ≈
for all R ∈ T do Union⁺(R) end for
while T ≠ ∅ do
    remove some set R from T
    for all a ∈ A do
        R' ← ∅
        for all (s, e) ∈ R do R' ← R' ∪ {Find(h(a)s, e)} end for
        if |R'| > 1 then
            Union⁺(R')
            T ← T ∪ {R'}
        end if
    end for
end while
```

To prove the correctness and running time of the algorithm, one can combine Lemma 2 with arguments similar to those given in the correctness and running time proofs of the

Hopcroft-Karp equivalence test [7]. We first show that the relation induced by the final partition is left-stable:

▶ **Lemma 3.** *Let (s, e) and (t, f) be linked pairs of the same class upon termination, then, for each $a \in A$, the pairs $(h(a)s, e)$ and $(h(a)t, f)$ are in the same class as well.*

Proof. We write $\mathrm{Find}_i(s, e) = \mathrm{Find}_i(t, f)$ if (s, e) and (t, f) belong to the same class after the i-th iteration of the *while*-loop. The index ∞ is used to describe the situation upon termination.

Let i be minimal such that for some pairs $(s, e), (t, f)$ and a letter $a \in A$, we have $\mathrm{Find}_i(s, e) = \mathrm{Find}_i(t, f)$ and $\mathrm{Find}_\infty(h(a)s, e) \neq \mathrm{Find}_\infty(h(a)t, f)$. Note that $i > 0$ because otherwise, a set containing both (s, e) and (t, f) would be added to T during initialization. Hence, there exists a pair (s', e') with $\mathrm{Find}_{i-1}(s', e') = \mathrm{Find}_{i-1}(s, e)$ and a pair (t', f') with $\mathrm{Find}_{i-1}(t', f') = \mathrm{Find}_{i-1}(t, f)$ such that $\mathrm{Union}^+(R)$ is executed for some set $R \supseteq \{(s', e'), (t', f')\}$. By choice of i, we have $\mathrm{Find}_\infty(h(a)s, e) = \mathrm{Find}_\infty(h(a)s', e')$ and $\mathrm{Find}_\infty(h(a)t, f) = \mathrm{Find}_\infty(h(a)t', f')$. Since we add the set R to T in iteration i, the equality $\mathrm{Find}_\infty(h(a)s', e') = \mathrm{Find}_\infty(h(a)t', f')$ holds as well, and thus $\mathrm{Find}_\infty(h(a)s, e) = \mathrm{Find}_\infty(h(a)t, f)$, a contradiction. ◀

There is of course a dual statement for the pairs $(s \cdot h(a), e)$ and $(t \cdot h(a), f)$.

▶ **Theorem 4.** *Let F be the set of linked pairs of S. When Algorithm 1 terminates, the classes of the partition correspond to the conjugacy classes of F. Furthermore, the algorithm executes at most*

- $|F| - 1$ Union *operations and*
- $2\,|A|\,(|F| - 1)$ Find *operations.*

Proof. By Lemma 3, the relation induced by the final partition is left-stable and throughout the main algorithm, two classes are only merged when required to establish this property. Thus, the relation is the finest left-stable equivalence relation coarser than $\approx$ and, by Lemma 2, equivalent to the conjugacy relation.

The number of Union operations is bounded by $|F| - 1$ since each operation reduces the number of classes in the partitions by 1. Let $R_1, \ldots, R_k$ be the sets that are added to T during the execution of the algorithm. Whenever one of the sets R_i is inserted into T, $|R_i| - 1$ Union operations are executed. Thus, we have

$$\sum_{i=1}^{k} \big(|R_i| - 1\big) \leqslant |F| - 1.$$

When R_i is removed from T, exactly $|A| \cdot |R_i|$ Find operations are executed in the same iteration of the *while*-loop. The total number of Find operations is therefore bounded by

$$\sum_{i=1}^{k} |A| \cdot |R_i| \leqslant \sum_{i=1}^{k} |A| \cdot (2\,|R_i| - 2) \leqslant 2\,|A| \cdot (|F| - 1)$$

where the first inequality follows from the fact that each of the sets R_i contains at least two elements. ◀

A sequence of n Union- and m Find-operations can be performed in $\mathcal{O}(n + m \cdot \alpha(n))$ time where $\alpha(n)$ denotes the extremely slow-growing *inverse Ackermann function* [14]. Thus, when considering a fixed-size alphabet, the total running time of our algorithm is "almost linear" in the number of linked pairs.

5 Testing for strong recognition

Common decision problems, such as the *universality problem* or the *inclusion problem*, are easy in the case of strong recognition. In the context of weak recognition, the algorithm presented in this section is a powerful tool to answer a broad range of similar problems. Given a morphism $h\colon A^+ \to S$ onto a finite semigroup S and two sets of linked pairs $P, Q \subseteq S \times E(S)$, it can be used to check whether $[P] \subseteq [Q]$. In particular, it allows for testing whether the morphism strongly recognizes a language $L = [P]$ by first computing the closure Q of P under conjugation and then using the algorithm to test whether $[Q] \subseteq [P]$.

The algorithm maintains two sets $R, T \subseteq S \times S \times S$. The former keeps record of the elements that are added to T during the course of the algorithm. To simplify the presentation, we define $x \cdot a^{-1}$ to be the set of all elements $p \in S^1$ which satisfy the equation $p \cdot h(a) = x$.

Algorithm 2 Testing for strong recognition

```
initialize R and T with the set {(s, e, 1) | (s, e) ∈ P}
while T ≠ ∅ do
    remove some element (s, x, y) from T
    if x = 1 then return "[P] ⊈ [Q]" end if
    if (sx, yxyx) ∉ Q then
        for all a ∈ A, p ∈ x · a^{-1} do
            if (s, p, h(a)y) ∉ R then add (s, p, h(a)y) to R and to T end if
        end for
    end if
end while
return "[P] ⊆ [Q]"
```

The following technical Lemma is crucial for the correctness proof of the algorithm:

▶ **Lemma 5.** *Let $u, v \in A^+$ and let (s, e) and $(h(u), h(v))$ be linked pairs. Then uv^ω is contained in $[s][e]^\omega$ if and only if there exists a factorization $v = v_1 v_2$ such that $v_1 \neq \varepsilon$, $h(uv_1) = s$ and $h(v_2 v v_1) = e$.*

Proof. Let $v = a_1 a_2 \cdots a_n$ with $n \geqslant 1$ and $a_i \in A$. If uv^ω is contained in $[s][e]^\omega$, there exists a factorization $uv^\omega = u' v_1' v_2' \cdots$ such that $h(u') = s$ and $h(v_i') = e$ for all $i \geqslant 1$. Since u and v are finite words, there exist indices $j > i \geqslant 1$, powers $k, \ell \geqslant 1$ and a position $m \in \{1, \ldots, n\}$ such that $u' v_1' v_2' \cdots v_{i-1}' = u v^k a_1 a_2 \cdots a_m$ and $v_i' v_{i+1}' \cdots v_j' = a_{m+1} a_{m+2} \cdots a_n v^\ell a_1 a_2 \cdots a_m$. We set $v_1 = a_1 a_2 \cdots a_m$ and $v_2 = a_{m+1} a_{m+2} \cdots a_n$. Then $v_1 v_2 = v$,

$$\begin{aligned} h(uv_1) = h(uv^k a_1 a_2 \cdots a_m) &= h(u' v_1' v_2' \cdots v_{i-1}') = s e^{i-1} &&= s \text{ and} \\ h(v_2 v v_1) = h(a_{m+1} a_{m+2} \cdots a_n v^\ell a_1 a_2 \cdots a_m) &= h(v_i' v_{i+1}' \cdots v_j') &&= e^{j-i+1} = e. \end{aligned}$$

To prove the converse direction, consider the factorization $uv^\omega = uv_1(v_2 v v_1)^\omega$. ◀

To simplify the proofs of the following two Lemmas, we extend h to a monoid morphism $h^1\colon A^* \to S^1$ by setting $h^1(u) = h(u)$ for all $u \in A^+$ and $h^1(\varepsilon) = 1$.

▶ **Lemma 6.** *If the difference $[P] \setminus [Q]$ is non-empty, the algorithm returns "$[P] \not\subseteq [Q]$".*

Proof. By the closure properties of regular languages, we know that there exists a word $\alpha = u(a_1 a_2 \cdots a_n)^\omega \in [P] \setminus [Q]$. Let $s = h(u)$ and $e = h(a_1 a_2 \cdots a_n)$. Lemma 5 shows that we can assume without loss of generality that (s, e) is contained in P. We now prove by induction on

the parameter k that upon termination, we have $(s, h^1(a_1a_2\cdots a_k), h^1(a_{k+1}a_{k+2}\cdots a_n)) \in R$ for all $k \in \{0,\dots,n\}$. In particular, by considering the case $k = 0$, we see that the element $(s, 1, e)$ is added to R. Since every element added to R is also added to Q, the algorithm returns "$[P] \not\subseteq [Q]$".

The base case $k = n$ is covered by the initialization of the set R. Let now $k < n$, $x = h^1(a_1a_2\cdots a_{k+1})$ and $y = h^1(a_{k+2}a_{k+3}\cdots a_n)$. By the induction hypothesis, we know that the tuple (s, x, y) is added to T during the course of the algorithm. Consider the iteration when this tuple is removed from T. Because of $\alpha \notin [Q]$, we know that $(sx, yxyx) \notin Q$. Thus the inner loop guarantees that $(s, h^1(a_1a_2\cdots a_k), h^1(a_{k+1}a_{k+2}\cdots a_n))$ is added to R. ◀

▶ **Lemma 7.** *If the algorithm returns "$[P] \not\subseteq [Q]$", the difference $[P] \setminus [Q]$ is non-empty.*

Proof. We construct a word in the difference $[P] \setminus [Q]$. For every triple $(s, e, 1)$ that is added to R during the initialization, we define $w[s, e, 1] = \varepsilon$. If a triple $(s, p, h(a)y)$ is added to R later, we set $w[s, p, h(a)y] = a \cdot w[s, p \cdot h(a), y]$. For every $(s, x, y) \notin R$, the word $w[s, x, y]$ is undefined. If $w[s, x, y]$ is defined, its image under h^1 is y and we have $(s, xy) \in P$. Both properties are easy to prove by induction.

Let $(s, 1, y)$ be the triple that was removed from T immediately before the termination of the algorithm. Consider an arbitrary word $u \in [s]$ and set $v = w[s, 1, y]$. We have $(s, y) \in P$ and thus $uv^\omega \in [P]$. For every factorization $v = v_1av_2$ where $v_1, v_2 \in A^*$ and $a \in A$, the word $w[s, h^1(v_1), h^1(av_2)]$ is defined as av_2 and thus, the tuple $(h(uv_1a), h(v_2vv_1a))$ is not contained in Q. In view of Lemma 5, this shows that $uv^\omega \notin [Q]$. ◀

We are now able to state the main result of this section:

▶ **Theorem 8.** *Given a morphism $h\colon A^+ \to S$ onto a finite semigroup S and two sets of linked pairs $P, Q \subseteq S \times E(S)$, one can check in $\mathcal{O}(|A| \cdot |S|^3)$ time whether $[P] \subseteq [Q]$.*

Proof. The correctness of Algorithm 2 follows from the previous two Lemmas. Since R contains at most $(|S|+1)^3$ elements when the algorithm terminates, the outer loop is executed at most $(|S|+1)^3$ times. Moreover, for all $a \in A$ and $s, t \in S$ with $s \neq t$, the sets $s \cdot a^{-1}$ and $t \cdot a^{-1}$ are disjoint. Thus, each element $p \in S^1$ is considered at most $|A| \cdot (|S|+1)^2$ times in the inner loop. If R is implemented as a bit field and T is implemented as a linked list, all operations take constant time. This shows that the total running time is in $\mathcal{O}(|A| \cdot |S|^3)$. ◀

6 Computation of the syntactic morphism

In this section, we present an algorithm to compute the syntactic semigroup for a given language. The syntactic homomorphism is obtained as a byproduct. One can show that the syntactic semigroup is the smallest semigroup strongly recognizing a language [1, 9], so this operation is in some sense analogous to minimization of finite automata. The most important difference is that our algorithm requires only quadratic time, whereas minimization is PSPACE-hard in the case of Büchi automata [8, 12].

Let S be a finite semigroup, let $h\colon A^+ \to S$ be a surjective morphism and let P be a set of linked pairs that is closed under conjugation. To make the following notation more readable, we define Q to be the maximal subset of $S \times S$ such that $[P] = [Q]$.

▶ **Lemma 9.** *Let $u, v \in A^+$. Then $uv^\omega \in [P]$ if and only if $(h(u), h(v)) \in Q$.*

Proof. Suppose that $uv^\omega \in [P]$. By Proposition 1 we have $[h(u)][h(v)]^\omega \subseteq [P] = [Q]$. Since Q is maximal, the pair $(h(u), h(v))$ is contained in Q. The converse implication is trivial. ◀

We now define a equivalence relation $\cong$ on S by $s \cong t$ if for all $z \in S$, we have

$$(z, s) \in Q \Leftrightarrow (z, t) \in Q \text{ and}$$
$$(s, z) \in Q \Leftrightarrow (t, z) \in Q.$$

Moreover, let $\equiv$ be the coarsest congruence on S that refines $\cong$, i.e., $s \equiv t$ if $xsy \cong xty$ for all $x, y \in S^1$. We denote by $[s]_\equiv$ the equivalence class $\{t \in S \mid t \equiv s\}$ of an element $s \in S$. The relation $\equiv$ is closely related to the syntactic congruence, as confirmed by the following result:

▶ **Proposition 10.** *The quotient semigroup $S/\equiv$ is isomorphic to $A^+/\equiv_L$.*

Proof. We first define a morphism $g\colon A^+ \to S/\equiv$ by setting $g(u) = [h(u)]_\equiv$ for all $u \in A^+$. Let now $u, v \in A^+$. By Lemma 9, we have $h(u) \equiv h(v)$ if and only if $h_L(u) = h_L(v)$. Thus, $g \circ h_L^{-1}$ is a semigroup isomorphism. ◀

The computation of the syntactic semigroup requires two steps:

1. Compute the partition induced by the equivalence relation $\cong$.
2. Refine the partition until the underlying equivalence relation becomes a congruence.

The first step can be performed in time quadratic in the size of the semigroup. For the second step, we can adapt Hopcroft's minimization algorithm for finite automata [6]. For $C \subseteq S$ and $a \in A$, we define

$$C \cdot a^{-1} = \{s \in S \mid s \cdot h(a) \in C\} \quad \text{and} \quad a^{-1} \cdot C = \{s \in S \mid h(a) \cdot s \in C\}.$$

The full algorithm is shown in Algorithm 3. It relies on the Split routine that is usually implemented as part of a *partition refinement data structure*, see e.g. [6] for details. Its semantics is shown in Algorithm 4. In addition to modifying the partition, that routine also updates a set $T \subseteq 2^S$ that is used in the main algorithm.

Algorithm 3 Computing the syntactic semigroup

```
initialize a partition with a single class S
for all s ∈ S do
    Split({t ∈ S | (s,t) ∈ Q})
    Split({t ∈ S | (t,s) ∈ Q})
end for
initialize T with the non-trivial classes of the partition
while T ≠ ∅ do
    remove some set C from T
    for all a ∈ A do
        Split(C · a^{-1})        ▷ Refine the partition and update T
        Split(a^{-1} · C)        ▷ Refine the partition and update T
    end for
end while
```

The next Lemma shows that upon termination, the equivalence relation induced by the partition is indeed a congruence:

▶ **Lemma 11.** *If, upon termination, the elements s and t belong to the same class of the partition, then, for each $a \in A$, the elements $h(a)s$ and $h(a)t$ are in the same class as well.*

Algorithm 4 The Split operation to refine a partition $\mathcal{P}$

```
procedure Split(X)
    for all C ∈ P do
        C1 ← C ∩ X, C2 ← C \ X
        if C1 ≠ ∅ and C2 ≠ ∅ then
            P ← (P \ {C}) ∪ {C1, C2}
            if C ∈ T then
                T ← (T \ {C}) ∪ {C1, C2}
            else
                if |C1| ⩽ |C2| then T ← T ∪ {C1} else T ← T ∪ {C2} end if
            end if
        end if
    end for
end procedure
```

Proof. Suppose that $h(a) \cdot s$ and $h(a) \cdot t$ belong to different classes. These elements are split either during the initialization or in the main loop. In either case, a set C that contains either $h(a) \cdot s$ or $h(a) \cdot t$ is added to T. When this set is removed from T, the operation $\mathrm{Split}(a^{-1} \cdot C)$ asserts that s and t lie in different classes as well. A dual argument holds in the right-sided case. ◀

There is of course a dual statement for the elements $s \cdot h(a)$ and $t \cdot h(a)$.

▶ **Theorem 12.** *The syntactic morphism can be computed in $\mathcal{O}(|S|^2 + |A| \cdot |S| \log |S|)$ time.*

Proof. Let us first argue that Algorithm 3 is correct. The partition is initialized with the equivalence classes of $\cong$. A class is only split when it is necessary to restore the left-stability or right-stability. Upon termination, the relation induced by the partition is a congruence, as stated in Lemma 11. Thus, it is the coarsest congruence that refines $\cong$ and hence equivalent to $\equiv$.

For the analysis of the running time, we assume that the operation $\mathrm{Split}(X)$ can be implemented in time linear in $|X|$. Then the initialization clearly takes $\mathcal{O}(|S|^2)$ time. We denote by $C_1, \ldots, C_k$ the sets that are added to T during the course of the algorithm. Let $s \in S$ and let $n_s = \{i \mid 1 \leqslant i \leqslant k, s \in C_i\}$ be the number of sets C_i containing s. At any point in time, there is at most one set in T that contains s. If such a set C is removed from T and another set C' with $s \in C'$ is added to T at a later point in time, we have that $|C'| \leqslant |C|/2$. Thus, the inequality $n_s \leqslant \log |S|$ holds for all $s \in S$ and we have

$$\sum_{i=1}^{k} \sum_{a \in A} \left(\left| C_i \cdot a^{-1} \right| + \left| a^{-1} \cdot C_i \right| \right) = \sum_{s \in S, a \in A} \left(n_{s \cdot h(a)} + n_{h(a) \cdot s} \right) \leqslant 2\,|A| \cdot |S| \log |S|.$$

Consequently, the total running time of the *while*-loop is in $\mathcal{O}(|A| \cdot |S| \log |S|)$, assuming that T is implemented efficiently, e.g. as a linked list. ◀

If the alphabet A is fixed and the semigroup S becomes large, the running time is dominated by the initialization. However, the following result implies that the algorithm we presented is quite optimal.

▶ **Proposition 13.** *Let $k \in \mathbb{N}$ and let $\lambda \in \mathbb{R}$ be a strictly positive number. One cannot compute the syntactic morphism in time $\mathcal{O}(|A|^k \cdot |S|^{2-\lambda})$.*

Proof. We assume that there exists a deterministic algorithm and a constant $c \geqslant 1$, such that every input of size $n = |S|$ and $m = |A|/2$ can be minimized in time $T(n,m) \leqslant c \cdot m^k \cdot n^{2-\lambda}$. Since c, k and λ are constant, there exists an integer $m \in \mathbb{N}$ with $2^{\lambda m} > 16c \cdot m^k$. Consider an alphabet $A = \{1, \ldots, 2m\}$ satisfying this condition.

We define $A_1 = \{1, \ldots, m\}$ and $A_2 = \{m+1, \ldots, 2m\}$, $S_1 = (2^{A_1} \setminus \{\emptyset\}) \times \{\emptyset\}$ and $S_2 = \{\emptyset\} \times (2^{A_2} \setminus \{\emptyset\})$. The set $S = S_1 \cup S_2$ forms a semigroup with the multiplication

$$(X_1, X_2) \cdot (Y_1, Y_2) = \begin{cases} (\emptyset, X_2 \cup Y_2) & \text{if } X_1 = Y_1 = \emptyset \\ (X_1 \cup Y_1, \emptyset) & \text{otherwise} \end{cases}$$

Furthermore, let $h\colon A^+ \to S$ be defined by $h(a) = (\{a\}, \emptyset)$ for all $a \in A_1$ and $h(b) = (\emptyset, \{b\})$ for all $b \in A_2$. Let F be the set of linked pairs of S. It is easy to verify that $S_1 \times S_2 \subseteq F$. Moreover, two tuples $(s,e), (t,f) \in S_1 \times S_2$ are conjugate if and only if $(s,e) = (t,f)$. The number of conjugacy classes of S is at least $|S_1| \cdot |S_2| \geqslant 2^{m-1} \cdot 2^{m-1} = 4^{m-1}$. The size of S is $n = |S_1| + |S_2| \leqslant 2^m + 2^m = 2^{m+1}$.

Consider the execution of the algorithm on input h and $P = F$. Since $[P] = A^\omega$, the algorithm returns the trivial semigroup. We denote by $(s_1, e_1), (s_2, e_2), \ldots, (s_\ell, e_\ell)$ the sequence of linked pairs for which the algorithm checks whether $(s_i, e_i) \in P$. We have $\ell \leqslant T(n,m) \leqslant c \cdot m^k \cdot n^{2-\lambda} < 4c \cdot m^k \cdot 2^{2m-\lambda m} = 16c \cdot m^k \cdot 2^{-\lambda m} \cdot 4^{m-1} < 4^{m-1}$ and thus, there is a conjugacy class C such that $(s_i, e_i) \notin C$ for all $i \in \{1, \ldots, \ell\}$. Since the algorithm is deterministic, the execution sequence on input $Q = P \setminus C$ is the same, and the algorithm returns, again, the trivial semigroup consisting of one element. However, $[Q] \neq A^\omega$ and thus, the algorithm is incorrect. ◀

One can also show that, independent of the alphabet size, it is impossible to compute the syntactic morphism in time $\mathcal{O}(|S|^{2-\lambda})$. However, the proof is a bit more involved [4].

7 Language operations on morphisms

One of the merits of strong recognition is that complementation is easy. If a morphism $h\colon A^+ \to S$ onto a finite semigroup S strongly recognizes a language $L \subseteq A^\omega$, it also strongly recognizes the complement $A^\omega \setminus L$. As in the case of finite words, we can use *direct products* for unions and intersections.

Another operation on languages which is of particular interest when it comes to converting MSO formulas to strongly recognizing morphisms are so-called *length-preserving morphisms*. Suppose we are given alphabets A, B and a length-preserving morphism $\pi\colon A^+ \to B^+$, i.e., $\pi(a) \in B$ for all $a \in A$. We naturally extend this morphism to infinite words by setting $\pi(a_1 a_2 \cdots) = \pi(a_1)\pi(a_2)\cdots$ and to languages $L \subseteq A^\omega$ by setting $\pi(L) = \{\pi(\alpha) \mid \alpha \in L\}$.

▶ **Proposition 14.** *Let $\pi\colon A^+ \to B^+$ be a length-preserving morphism, let S be a finite semigroup and let $h\colon A^+ \to S$ be a surjective morphism that strongly recognizes a language $L \subseteq A^\omega$. Then there exist a semigroup T of size $2^{|S|}$ and a morphism $g\colon B^+ \to T$ that strongly recognizes $\pi(L)$.*

Proof. We first define T to be the set 2^S of all subsets of S and extend it to a semigroup by defining an associative multiplication $X \cdot Y = \{xy \mid x \in X, y \in Y\}$. The morphism $g\colon B^+ \to T$ is uniquely defined by $g(a) = h(\pi^{-1}(a))$ for all $a \in B$.

Let us now verify that g strongly recognizes $\pi(L)$. Consider a linked pair (s,e) and two infinite words $\alpha, \beta \in g^{-1}(s)(g^{-1}(e))^\omega$. By Proposition 1, it suffices to show that $\alpha \in \pi(L)$ implies $\beta \in \pi(L)$. If α is contained in $\pi(L)$, we can conclude by Ramsey's theorem that

Table 1 Experimental results for different parameter values

	φ_k			ψ_k			χ_k		
	$\lvert S\rvert$	$\lvert F\rvert$	$\lvert P\rvert$	$\lvert S\rvert$	$\lvert F\rvert$	$\lvert P\rvert$	$\lvert S\rvert$	$\lvert F\rvert$	$\lvert P\rvert$
$k=2$	4	5	1	12	15	10	7	14	11
$k=3$	8	22	1	43	50	41	11	26	15
$k=4$	16	74	1	148	163	146	17	61	30
$k=5$	32	232	1	539	570	537	41	227	85
$k=6$	64	710	1	1863	1926	1861	105	716	184

there exists a linked pair (t, f) of S with $t \in s$, $f \in e$ and $h^{-1}(t)(h^{-1}(f))^\omega \cap L \neq \emptyset$. By assumption, h strongly recognizes L and thus, we have $h^{-1}(t)(h^{-1}(f))^\omega \subseteq L$. Since we know that there exists an infinite word $uv_1v_2\cdots \in \pi^{-1}(\beta)$ such that $h(u) = t$ and $h(v_i) = f$ for all $i \geqslant 1$, this immediately yields $uv_1v_2\cdots \in L$ and hence $\beta \in \pi(L)$. ◀

8 Experimental results

In order to test the algorithms and constructions in practice, we implemented the conversion of MSO formulas into strongly recognizing morphisms. The constructions described in Section 7 are used to recursively convert the formulas, and all intermediate results are minimized using the algorithm from Section 6. For details on MSO logic over infinite words and its connexion to regular languages, we refer to [15, 16]. The conversion to strongly recognizing morphisms instead of Büchi automata has the advantage that all intermediate objects can be minimized efficiently. Table 1 shows the size of the computed syntactic semigroup S, the number of linked pairs F and the size of the accepting set P (which is closed unter conjugation) for the following three families of MSO formulas with parameter $k \geqslant 1$ and free second-order variables $X_{k+1} = X_1, X_2, \ldots, X_k$:

$$\varphi_k = \forall x \bigwedge_{i=1}^{k} \exists y\,(x < y \wedge y \in X_i)$$

$$\psi_k = \forall x \forall y\,(y = x+1) \to \bigwedge_{i=1}^{k} (x \in X_i \to y \in X_{i+1})$$

$$\chi_k = \forall x \bigwedge_{i=1}^{k} (x \in X_i \to \exists y\,(x < y \wedge (y \in X_{i-1} \vee y \in X_{i+1})))$$

All computations were made on a Intel Core i5-3320M with 4GiB of RAM. The execution time was less than three seconds for each formula.

9 Summary and Outlook

We described several algorithms for weakly recognizing morphisms and strongly recognizing morphisms over infinite words. Our tests indicate that strongly recognizing morphisms, when combined with the minimization algorithm presented in Section 6, are a practical alternative to automata-based models when it comes to deciding properties of MSO formulas.

Some of the algorithms leave room for optimization. In particular, it would be interesting to see whether there is a linear-time algorithm to compute conjugacy classes and whether the running time of the algorithm described in Section 5 can be improved to $\mathcal{O}(|A| \cdot |S^2|)$.

References

1 A. Arnold. A syntactic congruence for rational ω-languages. *Theoretical Comput. Sci.*, 39:333–335, 1985.

2 H. Calbrix, M. Nivat, and A. Podelski. Ultimately periodic words of rational ω-languages. In *MFCS 94, Proceedings*, volume 802 of *LNCS*, pages 554–566. Springer, 1994.

3 V. Diekert and P. Gastin. First-order definable languages. In J. Flum, E. Grädel, and T. Wilke, editors, *Logic and Automata: History and Perspectives*, Texts in Logic and Games, pages 261–306. Amsterdam University Press, 2008.

4 L. Fleischer and M. Kufleitner. Efficient Algorithms for Morphisms over Omega-Regular Languages. *CoRR*, abs/1509.06215, 2015.

5 V. Froidure and J.-E. Pin. Algorithms for computing finite semigroups. In F. Cucker and M. Shub, editors, *Foundations of Computational Mathematics*, pages 112–126. Springer, 1997.

6 J. Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. In Z. Kohavi and A. Paz, editors, *Theory of Machines and Computations*, pages 189–196. Academic Press, New York, 1971.

7 J. Hopcroft and R. Karp. A linear algorithm for testing equivalence of finite automata. Technical report, Dept. of Computer Science, Cornell Univ., December 1971.

8 A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *13th Annual Symposium on Switching and Automata Theory*, pages 125–129. IEEE Computer Society, 1972.

9 D. Perrin and J.-É. Pin. *Infinite words*, volume 141 of *Pure and Applied Mathematics*. Elsevier, 2004.

10 J.-P. Pécuchet. Varietés de semisgroupes et mots infinis. In *STACS 86*, volume 210 of *LNCS*, pages 180–191. Springer, 1986.

11 M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:114–125, 1959. Reprinted in E. F. Moore, editor, *Sequential Machines: Selected Papers*, Addison-Wesley, 1964.

12 A. P. Sistla, M. Y. Vardi, and P. L. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Comput. Sci.*, 49(2-3):217–237, 1987.

13 L. J. Stockmeyer. The complexity of decision problems in automata theory and logic. PhD thesis, TR 133, M.I.T., Cambridge, 1974.

14 R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, Apr. 1975.

15 W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 4, pages 133–191. Elsevier, 1990.

16 W. Thomas. Languages, automata and logic. In A. Salomaa and G. Rozenberg, editors, *Handbook of Formal Languages*, volume 3, Beyond Words, pages 389–455. Springer, Berlin, 1997.