PSPACE-complete problems for subgroups of free groups and inverse finite automata[1]

J.-C. Birget[2] S. Margolis[3] J. Meakin[4] P. Weil[5]

Abstract

We investigate the complexity of algorithmic problems on finitely generated subgroups of free groups. Margolis and Meakin showed how a finite monoid $\mathrm{Synt}(H)$ can be canonically and effectively associated with such a subgroup H. We show that H is pure (that is, closed under radical) if and only if $\mathrm{Synt}(H)$ is aperiodic. We also show that testing for this property of H is PSPACE-complete. In the process, we show that certain problems about finite automata which are PSPACE-complete in general remain PSPACE-complete when restricted to injective and inverse automata (with single accept state), whereas they are known to be in NC for permutation automata (with single accept state).

We are concerned with the solution and the complexity of algorithmic problems about finitely generated subgroups of free groups. Our main results are that the problem of deciding purity for a finitely generated subgroup of a free group is decidable, and that it is PSPACE-complete.

Our techniques rely largely on automata theory. We first show that there are polynomial-time reductions, in both directions, between finitely generated subgroups of the free group $FG(\Sigma)$ over the finite alphabet Σ, and inverse automata over the symmetrized alphabet $\Sigma \cup \Sigma^{-1}$. Here a finitely generated subgroup of $FG(\Sigma)$ is specified by a finite set of words in $(\Sigma \cup \Sigma^{-1})^*$, whose total length is the size of the input. An inverse automaton over $\Sigma \cup \Sigma^{-1}$ is a deterministic finite automaton with single accept state, in which each letter $a \in \Sigma$ labels an injective partial transformation of the state set, and such that a^{-1} labels the inverse transformation.

The correspondence between subgroups of free groups and inverse automata was introduced by Reidemeister [20] in order to give a simple proof of the Nielsen-Schreier theorem that subgroups of free groups are free. This correspondence is well-known in combinatorial group theory where it has been used to compute invariants of finitely generated subgroups of free groups, such as rank and index (see Section 2), as well as to prove general results, such as the Nielsen-Schreier formula on the rank of finite index subgroups of $FG(\Sigma)$, the residual finiteness of $FG(\Sigma)$, or M. Hall's result on the embedding of finitely generated subgroups of $FG(\Sigma)$ as free factors of subgroups of finite index. One should add however that combinatorial group theorists do not, in general, view this as a correspondence between subgroups of $FG(\Sigma)$ and certain finite automata, but rather as a correspondence with certain Σ-labeled finite graphs, or more precisely, with certain immersions over the bouquet of $|\Sigma|$ circles. For details on these questions, see Lyndon and Schupp [15], Margolis and Meakin [16], Stallings [24], Sims [23].

[1]The first three authors were supported by NSF Grant 92-03981. The fourth author was supported by GdR-PRC *AMI*. All four authors were supported by the Center for Communication and Information Sciences, University of Nebraska-Lincoln.

[2]`birget@cse.unl.edu`. Department of Computer Science – University of Nebraska-Lincoln – Lincoln, NE 68588, USA.

[3]`margolis@bimacs.cs.biu.ac.il`. Department of Computer Science – Bar Ilan University – 52900 Ramat Gan, Israel.

[4]`jmeakin@unlinfo.unl.edu`. Department of Mathematics – University of Nebraska-Lincoln – Lincoln, NE 68588, USA.

[5]`weil@litp.ibp.fr`. LITP, IBP – 4 place Jussieu – 75252 Paris Cedex 05, France.

Margolis and Meakin [16] exploited the automata-theoretic point of view on this correspondence by showing that the finite Σ-labeled graph $\mathcal{A}_H$ associated with a finitely generated subgroup H of $FG(\Sigma)$, viewed as an inverse automaton, is the minimal automaton of a certain submonoid of the free inverse monoid over Σ which is canonically associated with H.

It is a natural idea is to use the algebraic properties of the transition monoid of $\mathcal{A}_H$, a finite inverse monoid denoted $\mathrm{Synt}(H)$, to explore the properties of H. It is important to note in this respect that finitely generated subgroups of $FG(\Sigma)$ are rational subsets of $FG(\Sigma)$, but are not in general recognizable [6]. So we cannot expect to find a finite automaton recognizing H itself. In fact, it is known that H is recognizable if and only if it has finite index, and that its syntactic congruence is the equality otherwise [22]. So, $\mathcal{A}_H$ does not recognize H; it corresponds to H in a more subtle way.

Note that the monoids of the form $\mathrm{Synt}(H)$, being transition monoids of inverse automata, are monoids of partial one-to-one transformations closed under taking inverses. Therefore they belong to the class of inverse monoids, that is, the class of monoids in which for each element x there exists a unique x^{-1} such that $xx^{-1}x = x$ and $x^{-1}xx^{-1} = x^{-1}$. This class has been widely studied by algebraists (see Petrich [18]), but also in relation with the theory of formal languages [27, 9]. In this paper however, we do not make explicit use of the specific properties of inverse semigroups.

It is known that H has finite index if and only if $\mathrm{Synt}(H)$ is a group. This statement is reminiscent of the situation prevailing in rational language theory, where a correspondence was established between certain combinatorially defined properties of rational languages and certain algebraically specified properties of finite monoids (Eilenberg's variety theorem, see Eilenberg [10], Pin [19]). Ruyle [21] proved an analogue of Eilenberg's variety theorem in the context of the study of finitely generated subgroups of free groups. That result however only provides a formal framework for this correspondence, and specific instances remain to be identified.

The first new result reported here gives a non-trivial instance of Ruyle's correspondence and is the free-group analogue of Schützenberger's theorem on star-free languages. By definition, H is *pure* if and only if $x^n \in H$ (with $n > 1$) implies $x \in H$. This is equivalent to saying that the subgroup H is closed under radical, where the radical of H is the set $\sqrt{H} = \{x \mid x^n \in H \text{ for some } n \neq 0\}$. We show:

Theorem *A finitely generated subgroup H of $FG(\Sigma)$ is pure if and only if* $\mathrm{Synt}(H)$ *(or equivalently, $\mathcal{A}_H$) is aperiodic.*

An immediate consequence of this result and of the effectiveness of the computation of $\mathrm{Synt}(H)$ is that the purity of a finitely generated subgroup of $FG(\Sigma)$ is decidable.

A related concept is that of *p-purity*. A subgroup H is p-pure for a prime number p if and only if the following holds: $x^n \in H$, with n relatively prime to p, implies $x \in H$. We prove that H is p-pure if and only if every subgroup of $\mathrm{Synt}(H)$ is a p-group. Again, this implies that it is decidable whether a given subgroup is p-pure for a given prime p.

The second new result concerns the complexity of deciding (p-)purity; we prove:

Theorem *The problem of deciding purity or p-purity of a finitely generated subgroup H of $FG(\Sigma)$, and the aperiodicity problem for finite inverse automata are* PSPACE*-complete.*

The aperiodicity problem for arbitrary finite automata is known to be PSPACE-complete (Cho and Huynh [7]). In our proof, we refine Cho and Huynh's approach to the case of inverse automata. In analogy with Cho and Huynh's proof, we first show that the *intersection-emptiness problem is* PSPACE*-complete for injective and for inverse automata*; this result is also of independent interest. For arbitrary finite automata this was first proved by Kozen [13]. We also make use of Bennett's theorem on injective Turing machines [5].

It is particularly interesting that the intersection-emptiness problem for inverse finite automata is PSPACE-complete, just as it is for arbitrary finite automata. The analogous problem for permutation automata (with single accept state) is known to have a fast parallel solution (in NC) [3]. Thus, while they may appear algebraically close to groups, inverse monoids behave more like arbitrary monoids with respect to certain classical complexity questions.

The result is no less interesting from the point of view of group theory. Much attention has been devoted to the algorithmic problems that arise when studying free groups and their quotients; however, the study of the computational complexity of these questions is still in its infancy. Regarding finitely generated subgroups of free groups, it is known that the generalized word problem and the conjugacy problem can be solved in polynomial time. These problems are in fact complete for P via logspace reductions (Avenhaus and Madlener, [1], [2]; for refinements, see [25]). Thus the problem of deciding purity is one of the few provably hard decidable problems known (as of today) in combinatorial group theory.

1 Inverse automata

First we fix some notation. Let Σ be an *alphabet*, i.e. a finite set, and let Σ^* be the free monoid on Σ, that is, the set of all words on Σ. The empty word is denoted by 1. A *deterministic finite state automaton* over Σ (or simply an *automaton*) is a structure $\mathcal{A} = (Q, \Sigma, \delta, i, F)$ where Q is the finite set of states, $i \in Q$ is the initial state, $F \subseteq Q$ is the set of final states and $\delta \colon Q \times \Sigma \to Q$ is the (partial) transition function, usually denoted $\delta(q, a) = q \cdot a$. As usual, δ is extended to a (partial) function on $Q \times \Sigma^*$ by letting $q \cdot 1 = q$ and $q \cdot (ua) = (q \cdot u) \cdot a$ (if this is defined) for all $q \in Q$, $u \in \Sigma^*$ and $a \in \Sigma$. The *language recognized by* $\mathcal{A}$ is the set $L(\mathcal{A}) = \{u \in \Sigma^* \mid i \cdot u \in F\}$.

All the automata considered in this paper are deterministic.

If every letter $a \in \Sigma$ induces a partial one-to-one function on Q and if $|F| = 1$, we say that $\mathcal{A}$ is *injective*. Note that this is equivalent to the condition that the reverse of $\mathcal{A}$ (the automaton obtained by reversing all the arrows of $\mathcal{A}$) is deterministic.

Let Σ^{-1} be a disjoint copy of Σ, together with a bijection $a \mapsto a^{-1}$ from Σ to Σ^{-1}. This bijection is extended to $(\Sigma \cup \Sigma^{-1})^*$ by letting $1^{-1} = 1$, $(a^{-1})^{-1} = a$ for each $a \in \Sigma$, and $(a_1 \cdots a_n)^{-1} = a_n^{-1} \cdots a_1^{-1}$ for all $n \geq 2$, $a_i \in \Sigma \cup \Sigma^{-1}$. We say that an injective automaton $\mathcal{A} = (Q, \Sigma \cup \Sigma^{-1}, \delta, i, \{f\})$ over this symmetrized alphabet is *inverse* if

$$\text{for all } p, q \in Q \text{ and } a \in \Sigma \cup \Sigma^{-1}, \quad p \cdot a = q \text{ if and only if } q \cdot a^{-1} = p.$$

There is a canonical way to *inversify* an injective automaton $\mathcal{A} = (Q, \Sigma, \delta, i, \{f\})$. Indeed, there is a unique way to extend δ to $Q \times (\Sigma \cup \Sigma^{-1})$ to make $(Q, \Sigma \cup \Sigma^{-1}, \delta, i, \{f\})$ an inverse automaton, namely by letting, for each $p, q \in Q$ and each $a \in \Sigma$, $\delta(q, a^{-1}) = p$ if and only if $\delta(p, a) = q$. The inverse automaton $(Q, \Sigma \cup \Sigma^{-1}, \delta, i, \{f\})$ is denoted ${}^{\mathrm{inv}}\mathcal{A}$.

For clarity, when representing an inverse automaton $\mathcal{A}$ over $\Sigma \cup \Sigma^{-1}$, it is convenient to represent only the Σ-labeled edges, since the Σ^{-1}-labeled edges can be deduced immediately from them. This representation is called the *positive state graph* of $\mathcal{A}$. (See examples in the next section.)

We say that a word of $(\Sigma \cup \Sigma^{-1})^*$ is *group-reduced* if it contains no factor of the form aa^{-1} or $a^{-1}a$ for $a \in \Sigma$. For each $z \in (\Sigma \cup \Sigma^{-1})^*$, there exists a unique group-reduced word $\mathrm{red}(z)$ such that $z = \mathrm{red}(z)$ in the free group $FG(\Sigma)$; $\mathrm{red}(z)$ is obtained by iteratively removing from z all factors of the form aa^{-1} or $a^{-1}a$. Now let $\mathcal{A}$ be an inverse automaton. By definition, the transitions induced by aa^{-1} and $a^{-1}a$ fix the states in their respective domains. So we have:

Lemma 1.1 *Let $\mathcal{A}$ be an inverse automaton over the alphabet $\Sigma \cup \Sigma^{-1}$, and let $z \in (\Sigma \cup \Sigma^{-1})^*$. If $p \cdot z = q$ in $\mathcal{A}$, then also $p \cdot \mathrm{red}(z) = q$ in $\mathcal{A}$.*

2 Subgroups of the free group and automata

Let H be a finitely generated subgroup of $FG(\Sigma)$, specified by a finite set Y of words in $(\Sigma \cup \Sigma^{-1})^*$ such that $H = \langle Y \rangle$. We construct an inverse automaton (or more precisely the positive state graph of an inverse automaton) from Y in three steps.

Construction of $\mathcal{A}_H$ and Synt(H)

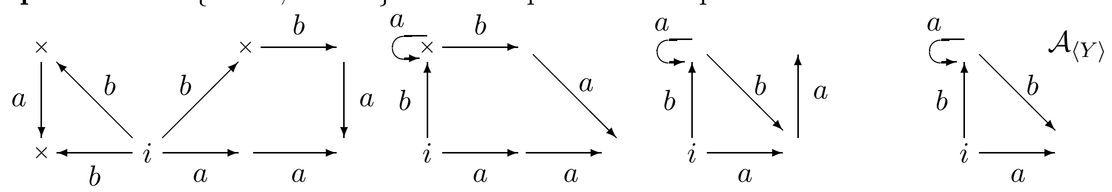
First, construct a set of $|Y|$ loops around a common distinguished vertex i, each labeled by an element of Y. By convention, since only Σ-labeled edges are indicated, an inverse letter a^{-1} in a word of Y gives rise to an a-labeled edge in the reverse direction on the corresponding loop.

Then, iteratively identify identically-labeled pairs of edges starting or ending at the same vertex. We now have the positive state graph of a connected inverse automaton, for which we let i be the unique initial and terminal state.

The last operation consists in “reducing” the automaton: iteratively remove from its state graph vertices of degree 1 other than i. In general, we say that an inverse automaton is *reduced* if in its positive state graph, no vertex has degree 1, except possibly its initial-terminal state.

It is known that the reduced inverse automaton $\mathcal{A}_H$ thus constructed is determined by H, not just by Y [24, 16]. The transition monoid of $\mathcal{A}_H$ is denoted by $\mathrm{Synt}(H)$.

Example 2.1 $Y = \{bab^{-1}, b^2aa^{-2}\}$. Some steps of the computation.



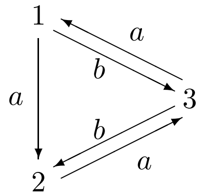
Conversely, given a connected reduced inverse automaton $\mathcal{A} = (Q, \Sigma \cup \Sigma^{-1}, \delta, i, \{i\})$, one can effectively construct a finite set of words Y such that $\mathcal{A} = \mathcal{A}_{\langle Y \rangle}$. Moreover this can be done in such a way that Y is a *basis* (of free generators) for $\langle Y \rangle = H$.

Construction of a basis Y

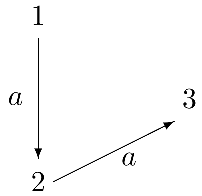
Let T be a spanning tree of the positive state graph Γ of $\mathcal{A}$. For each state q of $\mathcal{A}$, the tree T contains a unique shortest path from i to q: we let u_q be the label (in $(\Sigma \cup \Sigma^{-1})^*$) of this path. Let $p_j \xrightarrow{a_j} q_j$ $(1 \le j \le k)$ be the Σ-labeled edges of Γ which are not in T. For each j, let $y_j = u_{p_j} a_j u_{q_j}^{-1} \in (\Sigma \cup \Sigma^{-1})^*$, and let $H = \langle y_1, \ldots, y_k \rangle$. Then $\{y_1, \ldots, y_k\}$ is a basis for H and $\mathcal{A} = \mathcal{A}_H$ [24].

Example 2.2 Let $H = \langle a^2b^{-1}, ab^{-2}, ba, a^{-1}baba^{-1} \rangle$. Then $\{a^3, ba^{-2}, a^2ba^{-1}\}$ forms a basis of H.

$\mathcal{A}_H$



a spanning tree T



Before we proceed with the study of the properties of $\mathcal{A}_H$ let us observe that the two algorithms given above, to pass from a finitely generated subgroup of the free group to a connected reduced inverse automaton and back, are efficient.

Proposition 2.3 *Let Y be a finite subset of a free group and let n be the sum of the lengths of the words in Y. Let H be the subgroup generated by Y. Then $\mathcal{A}_H$ can be computed in time $O(n^2)$.*

Proof. It is clear that the collection of loops labeled by the elements of Y has n edges and $n - |Y| + 1$ vertices, and it can be computed in time proportional to n. Detecting a pair of edges to be identified can be done in time proportional to n. Since such identifications decrease the number of edges, there will be at most n identifications, so the second step of the algorithm can be implemented in time proportional to n^2. Finally, detecting and deleting extremal vertices of degree 1 can be done in time n. □

Remark 2.4 Notice that a group-reduced word is in H if and only if it labels a path from the initial-terminal state of $\mathcal{A}_H$ to itself. Since such a path can be detected in an automaton in time equal to the length of the word, Proposition 2.3 gives a linear algorithm for testing membership in a finitely generated subgroup of a free group if the generators are fixed. If the input includes a set Y of generators of H, then we have constructed an algorithm for the generalized word problem that runs in time $m + n^2$ where m is the length of the word to be tested and n is the sum of the lengths of words in Y. As mentioned in the introduction, this problem is complete for P.

Similarly, we have

Proposition 2.5 *Let $\mathcal{A}$ be a connected reduced inverse finite automaton. Then we can compute a basis for a subgroup H of the free group such that $\mathcal{A} = \mathcal{A}_H$ in polynomial time.*

The correspondence between finitely generated subgroups of the free group and finite automata has been explored early [15, 24], and it can be used notably to effectively compute invariants of such subgroups, such as rank, basis and index.

It follows from the constructions given above that if $H = \langle Y \rangle$ is a finitely generated subgroup of $FG(\Sigma)$, then the rank of H can be read directly from the positive state graph Γ of $\mathcal{A}_H$. Let v be the number of states of $\mathcal{A}_H$ and let e be the number of edges in Γ, that is, the number of Σ-labeled transitions of $\mathcal{A}_H$. Then any spanning tree of Γ has $v - 1$ edges, so that the rank of H is $e - v + 1$. Moreover, a basis for H can be computed as described above.

Another important and well-known property of H which can be read directly from $\mathcal{A}_H$ is the index of H in $FG(\Sigma)$. H has finite index if and only if $\mathcal{A}_H$ is a complete automaton over $\Sigma \cup \Sigma^{-1}$. In that case, $[FG(\Sigma) : H] = v$. At this point, it is easy to recover the Nielsen-Schreier formula relating the rank and index of a finite-index subgroup of $FG(\Sigma)$:

$$\text{if } H \text{ has finite index in } FG(\Sigma), \text{ then } \operatorname{rank}(H) - 1 = (|\Sigma| - 1)[FG(\Sigma) : H].$$

Since a complete inverse automaton is a permutation automaton (i.e., each letter induces a permutation of the states), the following properties of a finitely generated subgroup H of $FG(\Sigma)$ are equivalent [16, Theorem 5.1]:

(a) $\mathrm{Synt}(H)$ is a group.

(b) The inverse automaton $\mathcal{A}_H$ is a permutation automaton.

(c) H has finite index in $FG(\Sigma)$.

It follows immediately from this and Proposition 2.3 that given a finite subset Y of a free group, we can test if the subgroup H generated by Y is of finite index in time $O(n^2)$ where n is the sum of the lengths of words in Y.

This result is a model for the results we look at in this paper: a natural property of finitely generated subgroups of free groups can be detected by properties of both the automaton $\mathcal{A}_H$ and the monoid $\mathrm{Synt}(H)$. This leads to decidability and complexity results. Note that in the case of subgroups of finite index, we can detect this property by examining the graph of the automaton $\mathcal{A}_H$ and this leads to a polynomial-time algorithm for this problem. If we are forced to look at the structure of $\mathrm{Synt}(H)$, then there may be no quick algorithm, since the cardinality of $\mathrm{Synt}(H)$ may be exponential in the size of the input data for the subgroup H. We will see below that in the case of testing for purity, we cannot avoid this problem: we prove that purity is detected by an algebraic property of $\mathrm{Synt}(H)$, and that this problem is PSPACE-complete.

3 Pure subgroups and finite monoids

A finite monoid M is said to be *aperiodic* if it contains no non-trivial groups; equivalently, there exists $n \geq 1$ such that for all $x \in M$, we have $x^n = x^{n+1}$ (see [10, 19]). It is not difficult to see that n can always be chosen to be less than or equal to $|M|$. We say that a deterministic automaton is *aperiodic* if its transition monoid is aperiodic. This is equivalent to the following property:

$$\text{for each word } w, \text{ for each state } q \text{ and for each integer } n > 1, \quad q \cdot w^n = q \implies q \cdot w = q.$$

Finally we say that a subgroup H of a group G is *pure* (or closed under radical) if for each $x \in G$ and $n > 1$, $x^n \in H$ implies $x \in H$. Our characterization theorem is the following.

Theorem 3.1 *Let Σ be a finite alphabet and let H be a finitely generated subgroup of $FG(\Sigma)$. Then H is pure if and only if* $\mathrm{Synt}(H)$ *is aperiodic, if and only if $\mathcal{A}_H$ is aperiodic.*

Proof. Let i be the initial-terminal state of $\mathcal{A}_H$. First assume that H is pure, and assume that $q \cdot w^n = q$ in $\mathcal{A}_H$ for some state q of $\mathcal{A}_H$ and for some $w \in (\Sigma \cup \Sigma^{-1})^*$ and $n > 1$. Since $\mathcal{A}_H$ is connected, there exists $u \in (\Sigma \cup \Sigma^{-1})^*$ such that $i \cdot u = q$, so that $i \cdot (uwu^{-1})^n = i$. Therefore $(uwu^{-1})^n \in H$. But H is pure, so $uwu^{-1} \in H$. Therefore $i \cdot uwu^{-1} = i$, that is, $q \cdot w = q$. Thus, $\mathcal{A}_H$ is aperiodic.

Conversely, let us assume that $\mathcal{A}_H$ is aperiodic, and let $x \in FG(\Sigma)$, $n > 1$ be such that $x^n \in H$. Viewing x as a group-reduced word, we can factor it as $x = uwu^{-1}$ where w is a cyclically reduced word, that is, the powers of w are all reduced words. Since $x^n \in H$ and $\mathrm{red}(x^n) = uw^nu^{-1}$, we have $i \cdot uw^nu^{-1} = i$ and hence $(i \cdot u) \cdot w^n = i \cdot u$. By aperiodicity, it follows that $(i \cdot u) \cdot w = i$, that is, $i \cdot x = i \cdot uwu^{-1} = i$, and hence $x \in H$. Thus H is pure. $\square$

Example 3.2 Using the computations of Examples 2.1 and 2.2 above, it follows that $K = \langle bab^{-1}, b^2aa^{-1} \rangle$ is pure, while $H = \langle a^3, ba^{-2}, a^2ba^{-1} \rangle$ is not pure (a^3 labels a cycle in $\mathcal{A}_H$ while a does not).

For each prime number p, there is a corresponding notion of p-purity: we say that a subgroup H of a group G is *p-pure* if, for each $x \in G$ and for each integer $n > 1$ relatively prime to p, $x^n \in H$ implies $x \in H$. Like pure subgroups, p-pure subgroups of free groups are characterized by an algebraic property of the associated finite monoid.

Observe that if M is a finite monoid, then all subgroups of M are p-groups if and only if, for each $x \in M$, we have $x^i = x^{i+p^j}$ for some $i \geq 1$, $j \geq 0$. The translation of this property into a property of automata is as follows.

Proposition 3.3 *Let $\mathcal{A}$ be a finite automaton and let M be its transition monoid. The following conditions are equivalent:*

(a) *Every subgroup in M is a p-group.*

(b) *For each word w and for each state q of $\mathcal{A}$, if $q \cdot w^n = q$ for some n relatively prime to p, then $q \cdot w = q$.*

Proof. Let $\mu\colon \Sigma^* \to M$ be the transition morphism of $\mathcal{A}$. Then M acts on the set Q of states by $q \cdot (w\mu) = q \cdot w$ for each word w. That is, we can view M as a monoid of transformations of the set Q.

First let us assume that every subgroup of M is a p-group. Let $q \in Q$, $w \in \Sigma^*$ and $n \geq 1$ be such that $q \cdot w^n = q$ and n is relatively prime to p. Let $k \geq 1$ be minimal such that $q \cdot w^k = q$. Then the set of images of q under the iterated action of w is $\{q, q\cdot w, \ldots, q \cdot w^{k-1}\}$ and k divides n. In particular, k and p are relatively prime. Let $m = w\mu$. Since the set of images of q has k elements, there exists a morphism φ from the subsemigroup $\langle m \rangle$ generated by m onto the k-element cyclic group $\mathbf{Z}_k = \{0, 1, \ldots, k-1\}$, mapping m to 1. By the hypothesis on M, there exist integers $i \geq 1$, $j \geq 0$ such that $m^i = m^{i+p^j}$. Let us assume that j is minimal, and let $G = \{m^i, m^{i+1}, \ldots, m^{i+p^j-1}\}$. Now $G\varphi$ is a subgroup of $\mathbf{Z}_k$. If $j \neq 0$, then $G\varphi$ contains i and $i+1$, so it contains 1, and hence $G\varphi = \mathbf{Z}_k$. But this implies that k is a power of p, a contradiction. So $j = 0$, and we have $m^i = m^{i+\ell}$ for each ℓ. In particular, $m^{ik} = m^{ik+1}$, and hence $q \cdot w = (q \cdot w^{ik}) \cdot w = q \cdot w^{ik+1} = q \cdot w^{ik} = q$.

Conversely, let us assume that (b) holds, and let G be a subgroup of M with identity e. If G is not a p-group, then there exists an element $m \in G$ and an integer n such that n and p are relatively prime, $m \neq e$ and $m^n = e$. Since $em = m$ and $m^n = e$, the transformations e and m have the same domain. Let now q be a state in the domain of e. Then $(q\cdot e)\cdot m^n = (q \cdot e)\cdot e = q \cdot e$ since e is idempotent. So by the hypothesis, we have $(q \cdot e) \cdot m = q \cdot e$, which implies $q \cdot m = q \cdot e$ since $em = m$. So $e = m$, a contradiction. $\square$

A deterministic finite automaton which satisfies the two equivalent conditions of the above proposition will be called a *p-automaton*.

The next theorem is the analogue of Theorem 3.1.

Theorem 3.4 *Let H be a finitely generated subgroup of the free group $FG(\Sigma)$. Then H is p-pure if and only if every subgroup of* $\mathrm{Synt}(H)$ *is a p-group, if and only if $\mathcal{A}_H$ is a p-automaton.*

Proof. Let H be a finitely generated subgroup of $FG(\Sigma)$ and let i be the initial-terminal state of $\mathcal{A}_H$. Assume that H is p-pure. Let q be a state of $\mathcal{A}_H$ and assume that $q \cdot w^n = q$ for some $w \in (\Sigma \cup \Sigma^{-1})^*$ and for some n relatively prime to p. As in the proof of Theorem 3.1, we can find a word $u \in (\Sigma \cup \Sigma^{-1})^*$ such that $i \cdot uw^nu^{-1} = i$ and it follows that $(uwu^{-1})^n \in H$. Therefore, $uwu^{-1} \in H$ since H is p-pure. As in Theorem 3.1, we have $q \cdot w = q$ and thus every subgroup of $\mathrm{Synt}(H)$ is a p-group by Proposition 3.3.

Conversely assume that $\mathrm{Synt}(H)$ has the property that all its subgroups are p-groups. Suppose that $x^n \in H$ for some n relatively prime to p. Then, as in Theorem 3.1, we can write the reduced word x as $x = uwv$ where w is cyclically reduced, and we have $(i \cdot u) \cdot w^n = i \cdot u$. By Proposition 3.3, it follows that $(i \cdot u) \cdot w = i \cdot u$, that is, $i \cdot x = i$, $x \in H$. So H is p-pure. $\square$

Corollary 3.5 *Let Y be a finite subset of $G = FG(X)$. It is decidable whether the subgroup H generated by Y is pure, or p-pure for a given prime p.*

Proof. By the algorithm outlined in the previous section, we can compute $\mathcal{A}_H$ from the set Y. From this we can compute the multiplication table of $\mathrm{Synt}(H)$ from Y. It is clear that given the multiplication table for a finite monoid M we can effectively decide if every subgroup in M is trivial or a p-group for a given prime p. Therefore Theorem 3.1 and Theorem 3.4 give us algorithms to check for purity and p-purity respectively. □

Theorem 3.6 *Let Σ be a finite alphabet. The purity problem for finitely generated subgroups of the free group $FG(\Sigma)$, and the aperiodicity problem for inverse finite automata over Σ can be reduced to each other in polynomial time. Similarly, the p-purity problem for finitely generated subgroups of the free group $FG(\Sigma)$, and the p-automaton problem for inverse finite automata over Σ can be reduced to each other in polynomial time.*

Proof. The reduction from purity to aperiodicity is given in Proposition 2.3 and Theorem 3.1. The reduction from aperiodicity to purity is given in Proposition 2.5 and Theorem 3.1. The reduction from p-purity to the p-automaton problem is given in Proposition 2.3 and Theorem 3.4. The reduction from the p-automaton problem to p-purity is given in Proposition 2.5 and Theorem 3.4. □

In the next sections, we prove that the aperiodicity problem and the p-automaton problem for inverse finite automata are PSPACE-complete. These sections deal almost exclusively with automata, with very little reference to groups.

4 PSPACE-complete problems and injectiveness

Many fundamental problems about finite automata are PSPACE-complete (see e.g. [11]). In this paper, the intersection-emptiness problem and the aperiodicity problem for finite automata are particularly relevant. Recall that $L(\mathcal{A})$ denotes the language recognized by the automaton $\mathcal{A}$.

The intersection-emptiness problem Let Σ be a fixed finite alphabet of size at least 2.

INPUT A finite set $\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ of deterministic finite automata over the alphabet Σ; the finite automata are described by their transition tables; the number n is not fixed, but implicitly given as an input.

QUESTION $\bigcap_{1 \le i \le n} L(\mathcal{A}_i) = \emptyset$?

This problem was considered by Kozen, and proved to be PSPACE-complete [13]. We will see that the problem remains PSPACE-complete when various restrictions are imposed on the $\mathcal{A}_i$'s: the $\mathcal{A}_i$'s can be made *injective* and even *inverse* finite automata.

The aperiodicity problem Let Σ be a fixed finite alphabet of size at least 2.

INPUT A deterministic finite automaton $\mathcal{A}$ over the alphabet Σ.

QUESTION Is $\mathcal{A}$ aperiodic?

The problem was shown to be PSPACE-complete by Cho and Huynh [7] (Stern [26] had shown previously that it is in PSPACE and that it is co-NP-hard).

The next problem has not been considered in the literature. We prove in Section 7 that it is PSPACE-complete, even when the automata are restricted to being inverse.

The p-automaton problem Let Σ be a fixed finite alphabet of size at least 2, and let $p > 1$ be a fixed prime number.

INPUT A deterministic finite automaton $\mathcal{A}$ over the alphabet Σ.

QUESTION Is $\mathcal{A}$ a p-automaton ?

Note that for the above problems the alphabet Σ is fixed, and not part of the input. For the aperiodicity and the p-automaton problems we consider also the analogous problems where the alphabet Σ of the automaton is not fixed, but part of the input. We call these problems the *aperiodicity* (*p-automaton*) *problem with variable alphabet*.

Our main new result in the next sections is that the above problems remain PSPACE-complete when *inverse* finite automata are used as input, instead of arbitrary finite automata. Our proofs follow in outline the ones for the non-inverse case [13], [26], and [7]: the intersection-emptiness problem is reduced to the aperiodicity (and the p-automaton) problem; we have already reduced the latter to the purity (p-purity) problem.

We add two new ingredients:

- Bennett's remarkable theorem [5] about the space complexity of *injective* Turing machines; see also [14].

- A proof that the introduction of *inverses* into injective finite automata does not destroy PSPACE-completeness of the problems we consider.

In the study of space complexity it is enough to consider *one-tape* Turing machines only. We will need detailed notation in our later constructions. A one-tape Turing machine is a structure $(Q, \Gamma, \Sigma, \delta, q_0, q_f)$, where Q is the set of states, q_0 is the start state, q_f is the sole accept state, Γ is the total alphabet, Σ (a subset of Γ) is the input alphabet, and δ is the transition function. For each $q \in Q$, we let $\Gamma_q = \{\binom{a}{q} \mid a \in \Gamma\}$, and for each $X \subseteq Q$, we let $\Gamma_X = \bigcup_{q \in X} \Gamma_q$. Now we let $\Delta = \Gamma \cup \Gamma_Q$. Then Δ is the alphabet of the configurations of the Turing machine: a configuration can be viewed as a word over Δ with exactly one letter in Γ_Q (with the convention that, at any moment, the read-write head of the Turing machine is located on a given cell). If c and c' are configurations and c' is obtained from c by application of one transition of the Turing machine, then we write $c \vdash c'$.

The one-tape Turing machines considered here have further restrictions, that do not affect space complexity. We always assume that their start state and their accept state are distinct, i.e. $q_f \neq q_0$. They have two kinds of transitions: read-write transitions (in which the read-write head does not move), and shift transitions (in which nothing is printed on the tape, and nothing is read; depending on the current state only, a new state is entered and the head moves left or right).

Using the formalism of configurations, a read-write transition is of the form $\binom{a}{q} \to \binom{b}{p}$ with $a, b \in \Gamma$ and $p, q \in Q$. That is, the initial position of the read-write head is on a cell containing a and the machine is in state q; after the transition, the read-write head has not moved, the machine is now in state p, letter a has been erased from the tape, and letter b has been written instead.

A right-moving transition is of the form $\binom{a}{q}b \to a\binom{b}{p}$ with $a, b \in \Gamma$ and $p, q \in Q$. That is, the initial position of the read-write head is a cell containing a, to the right of which there is a cell containing b, and the state is q; after the move, the read-write head has moved one cell to the right (i.e. it is now on the cell containing b) and the machine is in state p; nothing has been written or erased on the tape. Similarly, a left-moving transition is of the form $a\binom{b}{q} \to \binom{a}{p}b$. Also, we require that the right (and left) moving transitions are "oblivious": if $\binom{a}{q}b \to a\binom{b}{p}$ is a transition then $\binom{x}{q}y \to x\binom{y}{p}$ is also a transition for every $x, y \in \Gamma$. In other words, a right (or left) moving transition will be triggered solely on the basis of the state of the machine, independently of the symbol read by the read-write head at that point.

In the one-tape Turing machines we consider, we will always assume that the start state q_0 is a source, i.e., it does not occur on the right side of any transition. We will also assume that the accept state q_f is a sink, i.e., it does not occur on the left side of any transition. Neither assumption affects space complexity.

We say that a state q is right-moving (or left-moving, or read-write) if q occurs on the left side of some right-moving (resp. left-moving, resp. read-write) transition. We denote the set of right-moving (resp. left-moving, resp. read-write) states by Q_r (or Q_ℓ, or Q_w). Since q_f is a sink, it is neither in Q_r, nor in Q_ℓ, nor in Q_w.

By definition, the above Turing machine is *deterministic* if the state set Q is partitioned as $Q = Q_r \cup Q_\ell \cup Q_w \cup \{q_f\}$, and if, for every read-write state $q \in Q_w$ and every $a \in \Gamma$, $\binom{a}{q}$ occurs in the left side of at most one transition. In particular, in every configuration, at most one transition is applicable.

We say that a state q is reached by a right-moving (or left-moving or read-write) transition if q occurs on the right side of some right-moving (resp. left-moving, resp. read-write) transition. By Q^r (or Q^ℓ or Q^w) we denote the set of states reachable by right-moving (resp. left-moving, resp. read-write) transitions. Since q_0 is a source, it is neither in Q^r, nor in Q^ℓ, nor in Q^w.

By definition, the above Turing machine is *injective* if the state set Q is partitioned as $Q = Q^r \cup Q^\ell \cup Q^w \cup \{q_0\}$, and if for every state $q \in Q^w$ and every $a \in \Gamma$, $\binom{a}{q}$ occurs in the right side of at most one transition. In particular, every configuration can be reached by at most one transition.

The following is part of Bennett's results:

Theorem 4.1 [5] *Let $L \subseteq \Sigma^*$ be a language which is recognized by a deterministic Turing machine with space-complexity $S(\cdot)$. Then L is also recognized by a (multi-tape) deterministic injective Turing machine with space-complexity $O(S(\cdot)^2)$, and with the property that when the machine halts all tapes are blank (except for the read-only input tape).*

Then we have:

Corollary 4.2 *Let $L \subseteq \Sigma^*$ be a language which is recognized by a deterministic Turing machine with space-complexity S(.); suppose also that $S(n) \geq \sqrt{n}$ for all n. Then L is also recognized by a deterministic injective one-tape Turing machine with space-complexity $O(S(.)^2)$, and with the following property:*

> *For every input $a_1a_2 \ldots a_{n-1}a_n \in \Sigma^*$, the start configuration is $a_1a_2 \ldots a_{n-1}\binom{a_n}{q_0}$, and the accept configuration (if $a_1 \ldots a_n$ is accepted) is $\binom{a_1}{q_f}a_2 \ldots a_{n-1}a_n$.*

Proof. We apply Bennett's theorem, and then the usual conversion of a multi-tape Turing machine into a one-tape machine (using the "tracks" idea); see e.g. [12, pp. 161–163]. This conversion preserves injectiveness, as is easy to check; it also preserves the space-complexity $O(S(\cdot)^2)$, provided $S(n)^2 \geq n$. The slightly unusual conventions about start and accept configurations will be useful later; note the symmetric appearance of the start and accept configurations. $\square$

Corollary 4.3 *There exists a* PSPACE*-complete language which is accepted by a deterministic injective one-tape Turing machine with space-complexity $S(n) = n$ for all n. For every input the time-complexity is an odd number. This Turing machine follows the same conventions as in Corollary 4.2, regarding the input configurations and the accept configurations; also q_0 is a source and q_f is a sink. Finally, the machine never visits the endmarkers of the tape.*

Proof. One starts with any PSPACE-complete language and applies Corollary 4.2. Next, one changes the language by padding the inputs, in order to obtain linear space. We can make sure that the endmarkers are never visited by using special letters at the ends of the input. This changes the language but does not affect PSPACE-completeness. $\square$

5 The intersection-emptiness problem

We are interested in the intersection-emptiness problem, the aperiodicity problem and the p-automaton problem when the automata are restricted to be inverse. In this section we show that the intersection-emptiness problem remains PSPACE-complete when the finite automata are injective or inverse. This result is also of independent interest.

In the rest of the paper, we fix an injective Turing machine $\mathcal{T} = (Q, \Gamma, \Sigma, \delta, q_0, q_f)$ with the properties described in Corollary 4.3. In particular, it is a deterministic injective one-tape Turing machine, which recognizes a PSPACE-complete language. The space complexity fonction is $S(n) = n$ for all n. Moreover, the computation of $\mathcal{T}$ on any input takes an odd number of steps. Also by hypothesis, q_0 is a source and q_f is a sink. $\Delta = \Gamma \cup \Gamma_Q$ is the alphabet of the configurations of $\mathcal{T}$. The initial configuration on any input, and the final configuration on any accepted input are as in Corollary 4.2. Finally, we let $\#$ be a new symbol not in Δ.

5.1 Injective automata

Proposition 5.1 *The intersection-emptiness problem for injective finite automata is* PSPACE-*complete.*

The intersection-emptiness problem for deterministic finite automata in general is PSPACE-complete (see Section 4), so its restriction to injective automata is in PSPACE.

Thus it suffices to reduce the Turing machine $\mathcal{T}$ (given above) to the intersection-emptiness problem for injective finite automata. The reduction is almost the same as in [13] (see also [7]), except that we must be careful, so that the injectiveness of the Turing machine leads to injective finite automata. The rest of Section 5.1 is devoted to this reduction.

Let $w = a_1 a_2 \cdots a_n \in \Sigma^*$ be an input for $\mathcal{T}$ of length n. With w we assign $2n$ finite automata $\mathcal{A}_i^0 = \mathcal{A}_i^0(w)$ and $\mathcal{A}_i^1 = \mathcal{A}_i^1(w)$, where $1 \leq i \leq n$. These automata play the same role as "$\mathcal{A}_i^{\text{even}}$" and "$\mathcal{A}_i^{\text{odd}}$" in [7], but they are constructed somewhat differently in order to be injective.

Let $c_0, c_1, \ldots, c_k$ be the sequence of configurations of the computation of $\mathcal{T}$ on input w; recall that k is assumed to be odd. The automata $\mathcal{A}_i^0$ and $\mathcal{A}_i^1$ $(1 \leq i \leq n)$ will be constructed so that

$$\bigcap_i L(\mathcal{A}_i^0) \cap L(\mathcal{A}_i^1) \;=\; \begin{cases} \{\#c_0\#c_1\#\cdots\#c_t\#\cdots\#c_k\#\#\} & \text{if } w \text{ is accepted by } \mathcal{T}, \\ \emptyset & \text{if } w \text{ is not accepted by } \mathcal{T}. \end{cases} \qquad (*)$$

For each i ($1 \leq i \leq n$), the finite automaton $\mathcal{A}_i^0$ checks whether for every even time index t, the positions $i-1$, i and $i+1$ in c_t and c_{t+1} are consistent with the requirement that $c_t \vdash c_{t+1}$ (where $0 \leq t < k$). The finite automaton $\mathcal{A}_i^1$ checks the same thing for every odd t; moreover, $\mathcal{A}_i^1$ checks that $c_0 = a_1 a_2 \cdots a_{n-1} \binom{a_n}{q_0}$ and that $c_k = \binom{a_1}{q_f} a_2 \cdots a_{n-1} a_n$. (As a result of the latter, we do not need the automaton $\mathcal{A}_{\text{ends}}$ of [7]. Also, the automaton $\mathcal{A}_{ID}$ of [7] is obviously redundant.)

When $i = 1$, the position $i - 1 = 0$ refers to the left endmarker of the Turing machine tape. Similarly, when $i = n$ then $i + 1$ refers to the right endmarker of the tape. By the construction of the Turing machine, these positions are actually never visited, hence $\mathcal{A}_1^0$, $\mathcal{A}_1^1$,$\mathcal{A}_n^0$ and $\mathcal{A}_n^1$ are a little simpler than the other $\mathcal{A}_i^0$ and $\mathcal{A}_i^1$ (when $1 < i < n$).

The detailed descriptions of $\mathcal{A}_i^1$ and $\mathcal{A}_i^1$ appear in Figures 1, 2 and 3. Each automaton has $O(n)$ states (more precisely, at most $|\Delta|^3 \cdot n$ states).

The state-graphs of $\mathcal{A}_i^1$ and $\mathcal{A}_i^0$ are self-explanatory except for the two regions called *diverging tree* and *converging tree*. Knowing what the automata are supposed to do, we can describe these trees and make sure that the finite automata are injective. Both trees are three edges deep when $1 < i < n$, and two edges deep when $i = 1$ or n.

Figure 3 describes the diverging tree and the converging tree, and how the two trees are connected together; the figure is for the case when $1 < i < n$. The figures for the cases when $i = 1$ or $i = n$ could easily be obtained from Figure 3 by leaving out level $i - 1$, resp. $n + 1$ (and discarding vertices that become disconnected this way).

The trees have $O(|\Gamma \times Q|)$ vertices, but most of these vertices play analogous roles; therefore, in the figures only a small number of vertices are given a vertex label.

In Figure 3, the edge $1 \xrightarrow{X} 2$ labeled by a subset X of the alphabet stands for the collection of all the transitions $1 \xrightarrow{x} 2$ labeled by the letters $x \in X$. Similarly, the edge $a \xrightarrow{\Gamma_{q_\ell}} (a, q_\ell, +)$ stands for the collection of all transitions $a \xrightarrow{\binom{x}{q_\ell}} (a, q_\ell, +)$ for $a, x \in \Gamma$ and $q_\ell \in Q_\ell$ (in particular these edges have different labels, different start vertices and different end vertices). This type of convention is used systematically in Figures 1, 2 and 3.

The vertex sets in the diverging and the converging trees are disjoint. Nevertheless, for typographical reasons, some different vertices are drawn with the same label in the figures (e.g., there are two vertices labeled "2" in the different trees; they are intended to be distinct). So, a rigorous definition of the vertex set of the diverging tree will be:

$$
\begin{aligned}
V_{\text{div}} &= \{div\} \times V' \qquad \text{where} \\
V' &= \{1,2\} \cup Q_r \cup (Q_r \times \Gamma) \cup (Q_r \times \Gamma \times \{+\}) \cup \Gamma_{Q_\ell} \cup (\Gamma_{Q_\ell} \times \{+\}) \\
&\quad \cup \Gamma_{Q_w} \cup (\Gamma_{Q_w} \times \{+\}) \cup \Gamma \cup (\Gamma \times \{+\}) \cup (Q_\ell \times \Gamma \times \{+\}) \\
&\quad \cup \Gamma_{Q_r} \cup (\Gamma_{Q_r} \times \{+\}).
\end{aligned}
$$

Note that we distinguish between $\binom{a}{q_r} \in \Gamma_{Q_r}$ and $(q_r, a) \in Q_r \times \Gamma$.

The definition of the vertex set of the converging tree is similar:

$$
\begin{aligned}
V_{\text{conv}} &= \{conv\} \times V'' \qquad \text{where} \\
V'' &= \{1,2\} \cup Q^r \cup (\Gamma \times Q^r) \cup (\{-\} \times \Gamma \times Q^r) \cup \Gamma_{Q^\ell} \cup (\{-\} \times \Gamma_{Q^\ell}) \\
&\quad \cup \Gamma_{Q^w} \cup (\{-\} \times \Gamma_{Q^w}) \cup \Gamma \cup (\{-\} \times \Gamma) \cup (\{-\} \times Q^\ell \times \Gamma) \\
&\quad \cup \Gamma_{Q^r} \cup (\{-\} \times \Gamma_{Q_r}).
\end{aligned}
$$

We omit the formal definition of the edge sets, since they are clear from Figure 3. D denotes

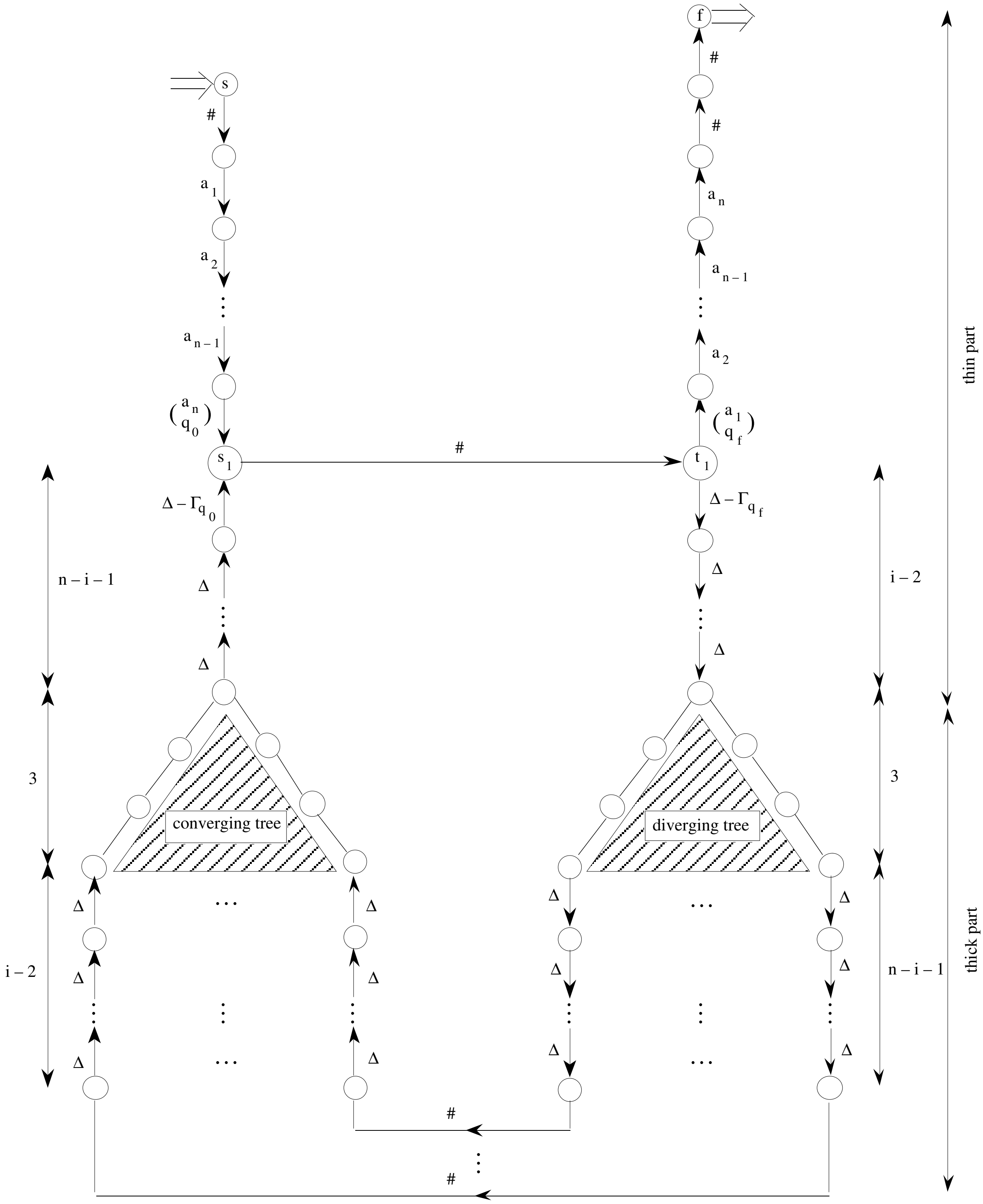


Figure 1: $\mathcal{A}_i^1$

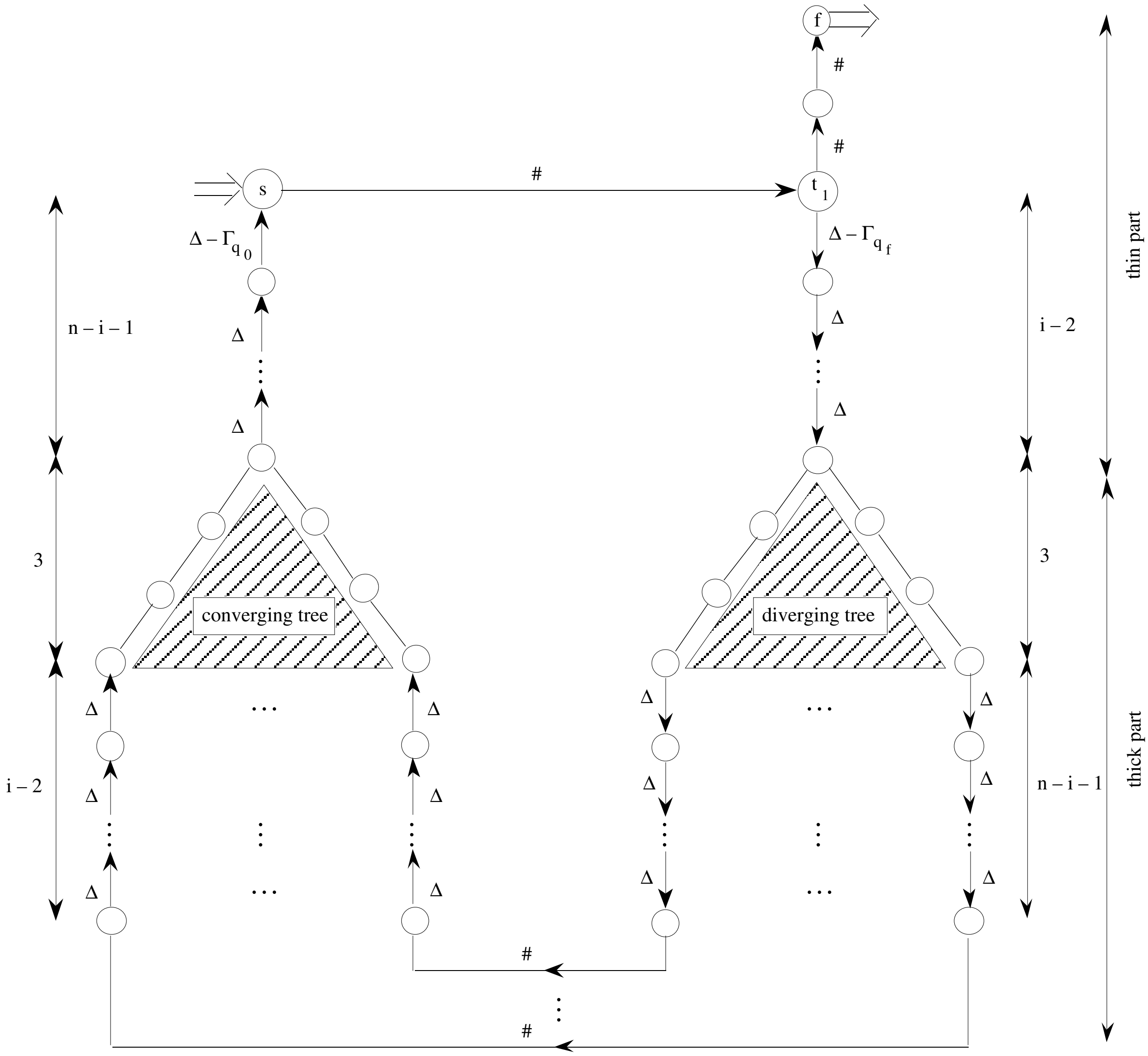


Figure 2: $\mathcal{A}_i^0$

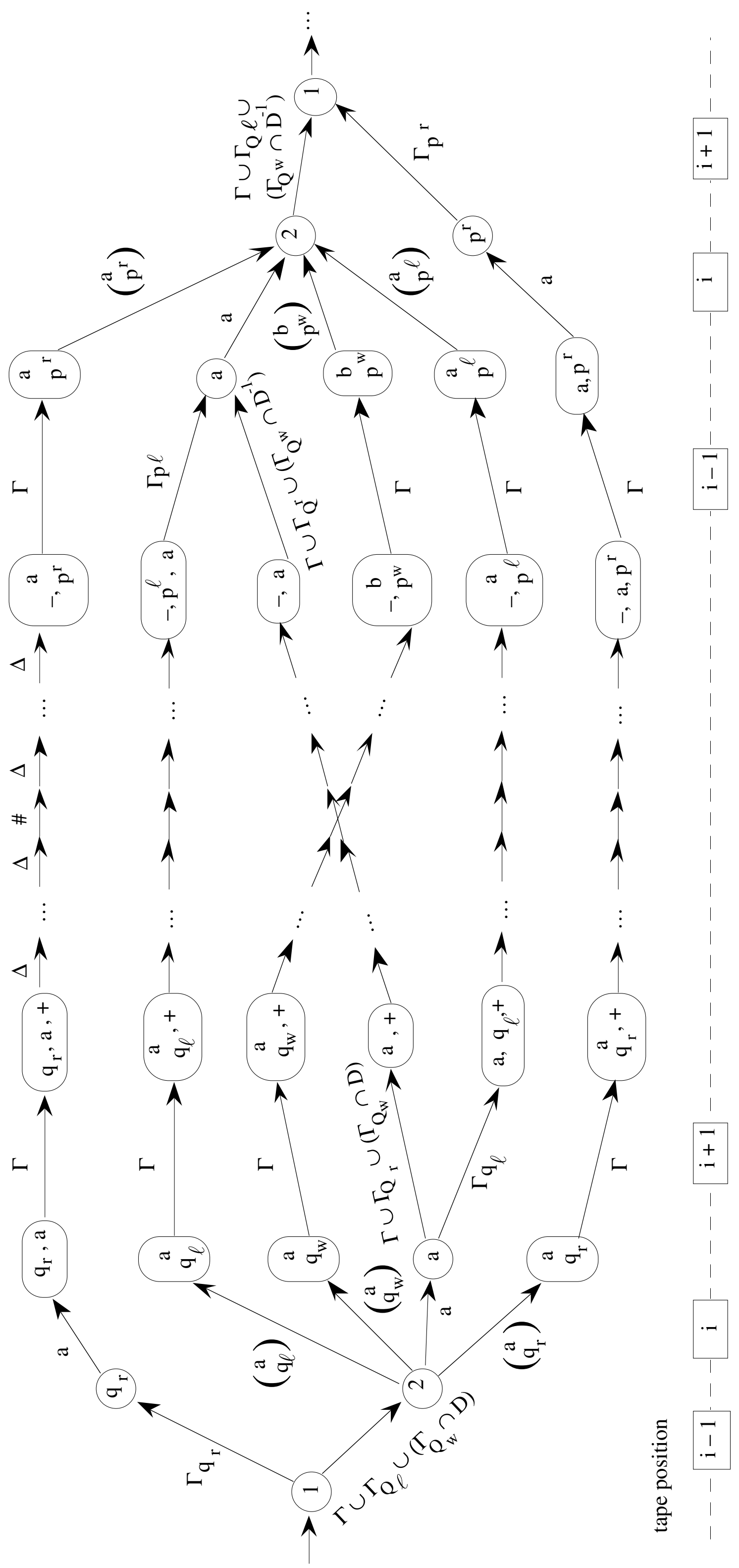


Figure 3: Diverging tree and converging tree, and their interconnection (in $\mathcal{A}_i^0$ and $\mathcal{A}_i^1, 1 < i < n$)

the set of left-hand sides of read-write transitions of the Turing machine, and D^{-1} denotes the set of right-hand sides of these transitions.

Finally, the diverging tree and the converging tree of $\mathcal{A}_i^1$ are interconnected by paths of length $n-2$, labeled by $\Delta^{n-i-1}\#\Delta^{i-2}$ (see Figure 3). The details of the interconnection are as follows:

Vertex $(q_r, a, +)$ in the diverging tree is connected to vertex $\left(-, \binom{a}{p^r}\right)$ in the converging tree if and only if the transition $\binom{x}{q_r}a \to x\binom{a}{p^r}$ exists in the Turing machine (and the existence of such a transition does not depend on $x, a \in \Gamma$, since right- and left-moves are oblivious). Also, $(q_r, a_1, +)$ is never connected to $\left(-, \binom{a_2}{p^r}\right)$ if $a_1 \neq a_2$.

Similarly, vertex $\left(\binom{a}{q_\ell}, +\right)$ is connected to vertex $(-, p^\ell, a)$ if and only if in the Turing machine $x\binom{a}{q_\ell} \to \binom{x}{p^\ell}a$; vertex $\left(\binom{a}{q_w}, +\right)$ is connected to vertex $\left(-, \binom{b}{p^w}\right)$ if and only if $\binom{a}{q_w} \to \binom{b}{p^w}$ in the Turing machine; vertex $(a, +)$ is connected to vertex $(-, a)$; vertex $(a, q_\ell, +)$ is connected to vertex $\left(-, \binom{a}{p^\ell}\right)$ if and only if $a\binom{x}{q_\ell} \to \binom{a}{p^\ell}x$ in the Turing machine; and $\left(\binom{a}{q_r}, +\right)$ is connected to vertex $(-, a, p^r)$ if and only if $\binom{a}{q_r}x \to a\binom{x}{p^r}$ in the Turing machine.

One can see from Figures 1, 2 and 3 that the automata $\mathcal{A}_i^0$ and $\mathcal{A}_i^1$ are all deterministic and injective, using the fact that Q_r, Q_ℓ, Q_w are disjoint, and Q^r, Q^ℓ, Q^w are disjoint (due to the determinism and injectiveness of the Turing machine). It is also clear that each $\mathcal{A}_i^0$ and $\mathcal{A}_i^1$ has $O(n)$ states, and has just one accept state.

Although most of the time we will write $\mathcal{A}_i^0$ or $\mathcal{A}_i^1$ (instead of $\mathcal{A}_i^0(w)$, $\mathcal{A}_i^1(w)$) we must keep in mind that these finite automata depend on $w = a_1 a_2 \cdots a_n$.

So far we have completed the definition of the injective finite automata $\mathcal{A}_i^0, \mathcal{A}_i^1$ $(1 \leq i \leq n)$. We still have to prove that these automata work as intended, i.e., that the PSPACE-complete language accepted by the Turing machine $\mathcal{T}$ reduces to the Intersection-Emptiness problem of the $\mathcal{A}_i^0$ and $\mathcal{A}_i^1$. This is done in the next Lemma.

Lemma 5.2 *The word w is accepted by the Turing machine $\mathcal{T}$ if and only if*

$$\bigcap_{1 \leq i \leq n} (L(\mathcal{A}_i^0) \cap L(\mathcal{A}_i^1)) \neq \emptyset.$$

Proof. In fact, we prove that Condition $(*)$ above holds. One direction is easy. Recall that $w = a_1 a_2 \cdots a_n$. If w is accepted by the $\mathcal{T}$, then there is an accepting computation $c_0 \vdash c_1 \vdash \cdots \vdash c_k$ where $c_0 = a_1 \cdots a_{n-1}\binom{a_n}{q_0}$ and $c_k = \binom{a_1}{q_f}a_2 \cdots a_n$. Then the string

$$\#c_0\#c_1\# \ldots \#c_t\#c_{t+1}\# \ldots \#c_k\#\#$$

is accepted by each $\mathcal{A}_i^1$ and $\mathcal{A}_i^0$ $(1 \leq i \leq n)$. The argument is straightforward and is similar to the reasoning in [13] and [7]. Thus, the above intersection is non-empty.

We now prove the converse. If a word x is accepted by all the $\mathcal{A}_i^0$ and $\mathcal{A}_i^1$ then x is of the form

$$\#c_0\#c_1\# \ldots \#c_k\#\#.$$

It is clearly visible on Figures 1, 2 and 3 that $c_0 = a_1 \cdots a_{n-1}\binom{a_n}{q_0}$, $c_k = \binom{a_1}{q_f}a_2 \cdots a_n$, the c_t are in Δ^*. Moreover, each c_t has length n and is a valid configuration (i.e. $c_t \in \Gamma^*\Gamma_Q\Gamma^*$). We now show that $c_t \vdash c_{t+1}$ for each t.

Assume, by induction, that $c_0, \ldots, c_t$ are well formed configurations and that they follow from each other in that order. Let $c_t = b_1 \ldots b_{i-1}\binom{b_i}{q}b_{i+1} \ldots b_n$ where $b_1, \ldots, b_n \in \Gamma$, $q \in Q$, and $1 \leq i \leq n$. Let $c_{t+1} = d_1 \ldots d_n$. We want to show that c_{t+1} is of the form $c_{t+1} =$

$b_1 \cdots b_{i-2}\ d_{i-1}\ d_i\ d_{i+1}\ b_{i-2} \cdots b_n$, where the three letters $d_{i-1}\ d_i\ d_{i+1}$ are determined by the three letters $b_{i-1}\binom{b_i}{q}b_{i+1}$ according to a transition of the Turing machine $\mathcal{T}$; in particular, one of the letters d_{i-1}, d_i, d_{i+1} should belong to Γ_Q and the other two should belong to Γ.

(1) Since the word $\#c_0 \ldots c_k\#\#$ is accepted by each $\mathcal{A}_j^0$ and $\mathcal{A}_j^1$ for $1 \le j \le i-2$ or $i+2 \le j \le n$, we have $b_j = d_j$ for all these values of j. Indeed, $b_{j-1}\ b_j\ b_{j+1}$ consists of three letters in Γ in this case, so when we read $b_{j-1}\ b_j\ b_{j+1}$ in the diverging tree, and follow its connection with the converging tree in $\mathcal{A}_j^1$ or $\mathcal{A}_j^0$ (Figure 3), we conclude that $b_j = d_j$ (otherwise the automata would not accept).

(2) Since $\#c_0 \ldots c_k\#\#$ is accepted by $\mathcal{A}_{i-1}^1$ and $\mathcal{A}_{i-1}^0$ with $b_{i-2}\ b_{i-1}\binom{b_i}{q}$ being read in the diverging tree (Figure 3), we conclude:
if $q \in Q_\ell$ and $a\binom{b_i}{q} \to \binom{a}{p^\ell}b_i$ is a transition of $\mathcal{T}$, then $d_{i-1} = \binom{b_{i-1}}{p^\ell}$;
if $q \in Q_r \cup Q_w$, then $d_{i-1} = b_{i-1}$.

(3) Acceptance by $\mathcal{A}_i^0$ and $\mathcal{A}_i^1$ with $b_{i-1}\binom{bi}{q}b_{i+1}$ being read in the diverging tree implies:
if $q \in Q_\ell$ and $a\binom{b_i}{q} \to \binom{a}{p^\ell}b_i$ is a transition of $\mathcal{T}$, then $d_i = b_i$ and $d_{i-1} \in \Gamma_{p^\ell}$;
if $q \in Q_w$ and $\binom{b_i}{q} \to \binom{e_i}{p^w}$ is a transition of $\mathcal{T}$, then $d_i = \binom{e_i}{p^w}$;
if $q \in Q_r$ and $\binom{b_i}{q}a \to b_i\binom{a}{p^r}$ is a transition of $\mathcal{T}$, then $d_i = b_i$ and $d_{i+1} \in \Gamma_{p^r}$.

(4) Acceptance by $\mathcal{A}_{i+1}^1$ and $\mathcal{A}_{i+1}^0$ with $\binom{b_i}{q}b_{i+1}\,b_{i+2}$ being read in the diverging tree implies:
if $q \in Q_r$ and $\binom{b_i}{q}a \to b_i\binom{a}{p^r}$ is a transition of $\mathcal{T}$, then $d_{i+1} = \binom{b_{i+1}}{p^r}$;
if $q \in Q_\ell \cup Q_w$ then $d_{i+1} = b_{i+1}$.

In the above reasoning we have tacitly assumed that $1 < i < n$. The cases $i = 1$ or $i = n$ are not significantly different from the above. So, in every case (whether q belongs to Q_ℓ or Q_r or Q_w), c_{t+1} follows from c_t by applying the appropriate transition of the Turing machine $\mathcal{T}$.

Since c_0 is the initial configuration when reading w, since $\mathcal{T}$ is deterministic and since c_k is the accepting configuration corresponding to input w, it follows that $c_0, c_1, \ldots, c_k$ is an accepting computation of $\mathcal{T}$ on input w. □

This concludes the proof of Proposition 5.1.

5.2 Inverse automata

Now we show that the introduction of inverses preserves the PSPACE-completeness of the problem.

Proposition 5.3 *The intersection-emptiness problem for inverse finite automata is* PSPACE-*complete.*

Recall that according to our definition, inverse automata have only one accept state. For permutation automata with one accept state it is known [3] that the intersection-emptiness problem is in NC.

Proof. We adapt the reduction of the previous subsection to inverse automata, by taking the inversification of the above injective partial finite automata. That is, we assign to each input w of the Turing machine $\mathcal{T}$ the inverse automata ${}^{\rm inv}\mathcal{A}_i^0$ and ${}^{\rm inv}\mathcal{A}_i^1$, for $i = 1, \ldots, n = |w|$. By the next lemma this is indeed a reduction. □

Lemma 5.4 $\bigcap_{i=1}^{n}(L(\mathcal{A}_i^0) \cap L(\mathcal{A}_i^1)) \neq \emptyset$ if and only if $\bigcap_{i=1}^{n}(L({}^{\text{inv}}\mathcal{A}_i^0) \cap L({}^{\text{inv}}\mathcal{A}_i^1)) \neq \emptyset$.

Remark This property does not hold for the intersection of inverse automata languages in general; it strongly relies on the interdependence of the particular automata $\mathcal{A}_i^0$ and $\mathcal{A}_i^1$ used here.

Proof. Since the inversified languages are included the non-inversified ones, the left-to-right implication is immediate.

Conversely, let z be a word in $\bigcap_{i=1}^{n} L({}^{\text{inv}}\mathcal{A}_i^0) \cap L({}^{\text{inv}}\mathcal{A}_i^1)$. By Lemma 1.1, we may assume z to be group-reduced. We now show that z lies in $(\Delta \cup \{\#\})^*$ (i.e., no inverse letters appear). This will imply that z is in fact accepted by all the injective (non-inversified) automata $\mathcal{A}_i^0$ and $\mathcal{A}_i^1$ $(i = 1, \ldots, n)$.

Let $c_0 = a_1 \cdots a_{n-1}\binom{a_n}{q_0}$ and $c_k = \binom{a_1}{q_f}a_2 \cdots a_n$.

The word z is accepted by ${}^{\text{inv}}\mathcal{A}_i^1$, for every i, and the state graph of ${}^{\text{inv}}\mathcal{A}_i^1$ between the start state and the state s_1 is linear (a path). Therefore, since z is group-reduced, the prefix of z that ${}^{\text{inv}}\mathcal{A}_i^1$ has read when it reaches s_1 is $\#a_1 \cdots a_{n-1}\binom{a_n}{q_0} = \#c_0$.

The next letter in z, following $\#c_0$, must be $\#$. Indeed, looking now at the diverging tree of ${}^{\text{inv}}\mathcal{A}_n^0$, the only other possible next letter is $\binom{a_n}{q_0}^{-1}$ (see Figure 2 with $i = n$, and Figure 3). This however would contradict the assumption that z is group-reduced. This shows that z has the prefix $\#c_0\#$.

In a similar way one sees that z has the suffix $\#c_k\#\#$. So z is of the form

$$z = \#c_0\#u_1v_1^{-1}u_2 \ldots u_jv_j^{-1}u_{j+1} \ldots v_{N-1}^{-1}u_N\#c_k\#\#$$

with the u_j and v_j in $(\Delta \cup \{\#\})^*$.

We now show that v_j is the empty string, for all $j = 1, \ldots, N-1$, so that $z \in (\Delta \cup \{\#\})^*$. We assume (by contradiction) that all v_j's that are written down here are non-empty; in particular, v_1 is non-empty.

Let b be the right-most letter of u_1 (or let $b = \#$ if u_1 is empty), and let d be the right-most letter of v_1. Since $z = \#c_0\#u_1v_1^{-1} \cdots u_N\#c_k\#\#$ is group-reduced, we must have $b \neq d$. However then, letting $i = |\#c_0\ \#u_1| \mod (2n+2)$, we find that ${}^{\text{inv}}\mathcal{A}_i^0$ and ${}^{\text{inv}}\mathcal{A}_i^1$ will reject $\#c_0\#u_1v_1^{-1} \cdots u_N\#c_k\#\#$; indeed, at a state corresponding to tape position i in the diverging tree, a letter $b \in \Delta$ cannot be followed directly by a letter $d^{-1} \in \Delta^{-1}$ unless $b = d$ (see Figure 3). Thus, z would be rejected, contrary to our assumptions.

So v_j must be empty, for all $j = 1, \ldots, N-1$. □

6 The aperiodicity problem

We now turn to the aperiodicity problem, for injective and inverse automata.

6.1 Injective automata

Proposition 6.1 *The variable-alphabet aperiodicity problem for injective finite automata is* PSPACE*-complete.*

Cho and Huynh's proof [7] that this problem is PSPACE-complete for finite automata in general, serves as the basic framework here too. But now we use our injective finite automata $\mathcal{A}_i^1 = \mathcal{A}_i^1(w)$ and $\mathcal{A}_i^0 = \mathcal{A}_i^0(w)$ $(1 \leq i \leq n = |w|)$ and we prove (in the next subsection) that the method still works in the presence of inverses. To make the paper more self-contained we repeat now the essential ideas of [7]. We will first modify the alphabet of our injective finite automata and obtain new automata $\mathcal{B}_i^0 = \mathcal{B}_i^0(w)$ and $\mathcal{B}_i^1 = \mathcal{B}_i^1(w)$ $(1 \leq i \leq n)$. We will show that these automata are still injective, and the intersection of their languages is empty if and only if the Turing machine $\mathcal{T}$ (the same machine we used throughout Section 5 and with reference to which the automata $\mathcal{A}_i^1$ and $\mathcal{A}_i^0$ are built) accepts w. Moreover, each $\mathcal{B}_i^1$ and $\mathcal{B}_i^0$ $(1 \leq i \leq n)$ is *aperiodic*. Next, we connect all these finite automata in a cycle to obtain a finite automaton $\mathcal{B} = \mathcal{B}(w)$ which is injective and which has the property that $\mathcal{B} = \mathcal{B}(w)$ is aperiodic if and only if the Turing machine $\mathcal{T}$ does not accept w. This will show that the aperiodicity problem for injective finite automata is PSPACE-complete.

The automata $\mathcal{A}_i^0$ and $\mathcal{A}_i^1$ $(i = 1, \ldots, n)$ are not aperiodic, as observed for the analogous automata in [7]. E.g., for any word $u \in \#\Gamma^n$ we have: in $\mathcal{A}_i^0$, $\delta(s, u^2) = s$ and $\delta(s, u) \neq s$; and in $\mathcal{A}_i^1$, $\delta(s_1, u^2) = s_1$ and $\delta(s_1, u) \neq s_1$. We will make them aperiodic by "marking" the letters of the alphabet by the "distance" as in [7]; this preserves injectiveness.

Let s be the start state of $\mathcal{A}_i^1$ (or $\mathcal{A}_i^0$) $(i = 1, \cdots, n)$, and let q be any state. The distance of q (denoted $dist(q)$) is the length of the shortest directed path from s to q, taken modulo $2n + 2$. It is taken to be an integer between 0 and $2n + 1$.

The finite automata $\mathcal{B}_i^1$ and $\mathcal{B}_i^0$ $(i = 1, \ldots, n)$ are defined as follows. We start with $\mathcal{A}_i^1$ and $\mathcal{A}_i^0$, and in every transition $q \xrightarrow{a} \delta(q, a)$, we replace the label a by $(a, dist(q))$. In other words, we "mark" the letters by the distance of the previous state.

Thus the alphabet of each $\mathcal{B}_i^1$ and $\mathcal{B}_i^0$ is $\Delta_{\mathcal{B}} = (\Delta \cup \{\#\}) \times \{0, 1, \ldots, 2n + 1\}$. Obviously, these new automata are injective, since $\mathcal{A}_i^1$ and $\mathcal{A}_i^0$ are injective.

The following result of Cho and Huynh, and its proof still hold (and we refer to [7] for the proof).

Lemma 6.2 *Each finite automaton $\mathcal{B}_i^1$ and $\mathcal{B}_i^0$ $(i = 1, \ldots, n)$ is aperiodic.*

Recall also that, just like $\mathcal{A}_i^0$ and $\mathcal{A}_i^1$, $\mathcal{B}_i^0$ and $\mathcal{B}_i^1$ depend on the input w of the Turing machine $\mathcal{T}$; so we should actually write $\mathcal{B}_i^0(w)$ and $\mathcal{B}_i^1(w)$.

Let P be the smallest prime number satisfying $2n \leq P$. By a classical fact of number theory ("Bertrand's Postulate") we have $P < 4n$.

We will now construct an automaton $\mathcal{B}$ by putting P components $\mathcal{B}_i^0$, $\mathcal{B}_i^1$ in a cycle, as in [7]. Since we only have $2n$ such components, we first need to take more identical copies of them: We extend our definition of $\mathcal{B}_i^0$ and $\mathcal{B}_i^1$ by letting

$\mathcal{B}_i^0 = \mathcal{B}_{i \bmod n}^0$ when $n < i \leq (P - 1)/2$,

$\mathcal{B}_i^1 = \mathcal{B}_{i \bmod n}^1$ when $n < i \leq (P + 1)/2$.

Here $i \bmod n$ is chosen in the range $\{1, \ldots, n\}$. This gives us P automata. We choose the states of these automata so that different automata have disjoint state sets. We use the notation $\mathcal{B}_i^0 = (Q_i^0, \Delta_{\mathcal{B}}, \delta_i^0, s^{(0,i)}, \{f^{(0,i)}\})$, for $1 \leq i \leq (P - 1)/2$, and similarly for $\mathcal{B}_i^1$ (with 0 replaced by 1).

We now construct the automaton $\mathcal{B} = \mathcal{B}(w) = (Q, \Delta_{\mathcal{B}} \cup \{\flat\}, \delta, s^{(0)}, \{s^{(0)}\})$ by taking a disjoint union of the P automata $\mathcal{B}_i^0$ and $\mathcal{B}_i^1$ and connecting them by means of a new letter $\flat$ in such a way that for each i, $\delta(f^{(0,i)}, \flat) = s^{(1,i)}$ and $\delta(f^{(1,i)}, \flat) = s^{(0,\ (i+1) \bmod (P-1)/2)}$ (we pick $(i + 1)$ mod $(P - 1)/2$ in the range $\{1, \ldots,\ (P - 1)/2\}$). See Figure 4.

The automata $\mathcal{B}_i^0$ and $\mathcal{B}_i^1$ are thus connected in a cycle in the following order: first $\mathcal{B}_1^0$, then $\mathcal{B}_1^1$, $\mathcal{B}_2^0$, $\mathcal{B}_2^1$, $\mathcal{B}_3^0$, etc. The cycle consists of P component automata, so after $\mathcal{B}_n^1$, we have again $\mathcal{B}_{n+1}^0 = \mathcal{B}_1^0$, $\mathcal{B}_{n+1}^1 = \mathcal{B}_1^1$, etc.

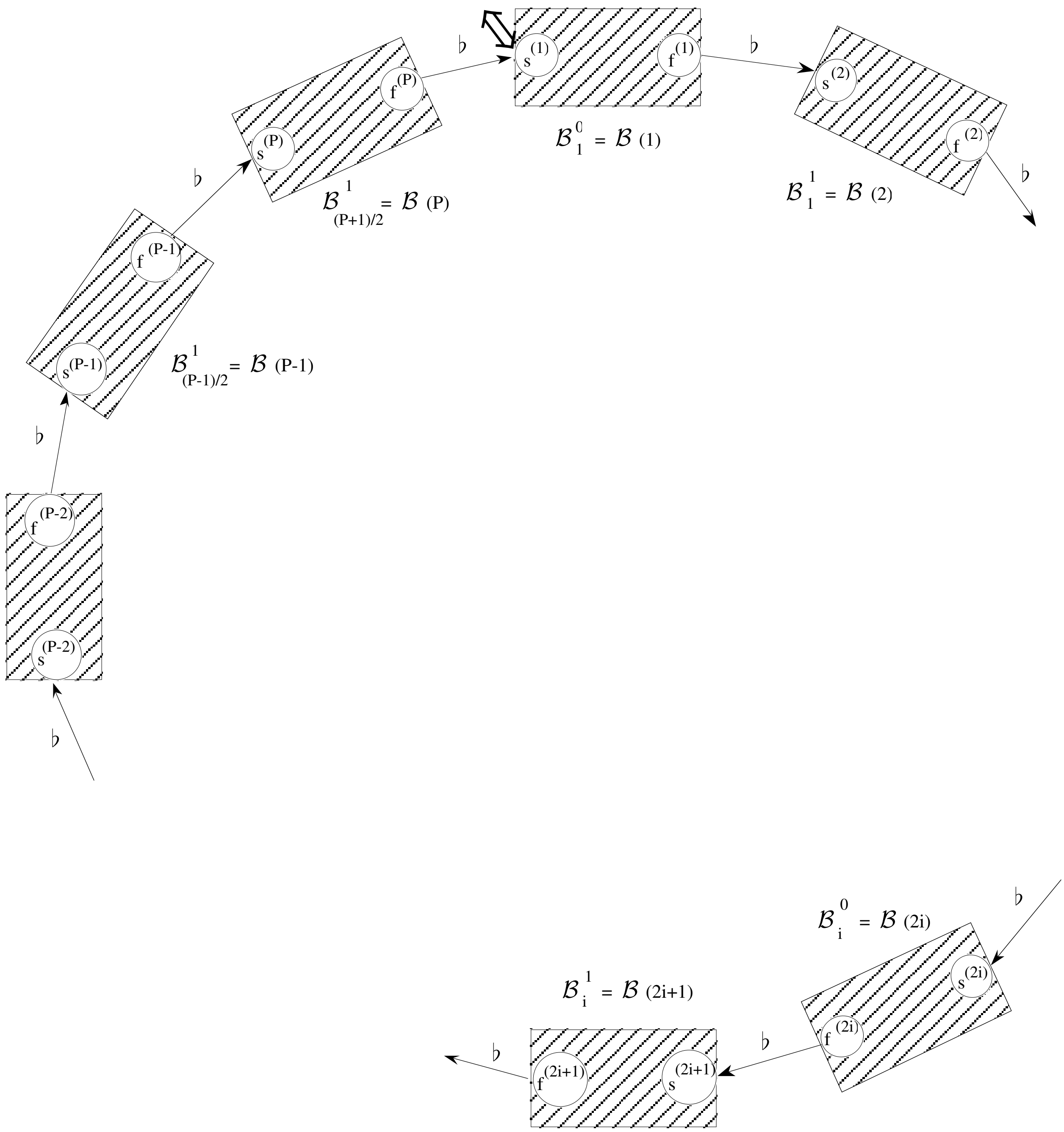


Figure 4: The automaton $\mathcal{B} = \mathcal{B}(w)$

Observe that the size of $\mathcal{B}$ is polynomial in $n = |w|$. The following result of Cho and Huynh [7], and its proof, still apply. (The proof of this lemma uses the primality of P. This primality is used again in the sequel, in Lemma 6.9.)

Lemma 6.3 *The finite automaton $\mathcal{B}(w)$ is aperiodic if and only if the Turing machine $\mathcal{T}$ does not accept the word w.*

Note that $\mathcal{B}$ is injective if each $\mathcal{B}_i^0$ and $\mathcal{B}_i^1$ is injective. All these automata use the alphabet $\Delta_{\mathcal{B}}$, whose size depends on n. This proves that the variable-alphabet aperiodicity problem for injective finite automata is PSPACE-hard. Now this problem is an instance of the aperiodicity problem for deterministic automata in general, and the latter problem is known to be PSPACE-complete. This concludes the proof of Proposition 6.1.

6.2 Inverse automata

Let us now extend this result for inverse automata.

Proposition 6.4 *The variable-alphabet aperiodicity problem for inverse automata is* PSPACE-*complete.*

We replace the previous injective finite automata $\mathcal{B}_i^0$, $\mathcal{B}_i^1$ and $\mathcal{B}$ by their inversifications. Then we show that the statements of Lemmas 6.2 and 6.3 still hold for these inversified finite automata. This is not straightforward; the inversification adds edges to the automata and this creates cycles which could conceivably destroy aperiodicity.

Lemma 6.5 *Each inverse finite automaton ${}^{\rm inv}\mathcal{B}_i^0$, ${}^{\rm inv}\mathcal{B}_i^1$ is aperiodic.*

Proof. In this proof we will simply write δ instead of ${}^{\rm inv}\delta_i^0$ or ${}^{\rm inv}\delta_i^1$ (the transition functions of ${}^{\rm inv}\mathcal{B}_i^0$ and ${}^{\rm inv}\mathcal{B}_i^1$ respectively). Let p be any state of ${}^{\rm inv}\mathcal{B}_i^0$ or ${}^{\rm inv}\mathcal{B}_i^1$, let $u \in (\Delta_{\mathcal{B}} \cup \Delta_{\mathcal{B}}^{-1})^*$, and let $m > 1$ be such that $\delta(p, u^m) = p$. Let $q = \delta(p, u)$. We want to show that $p = q$. By Lemma 1.1 we may assume that u is group-reduced.

We have $dist(p) = dist(q)$, because $\delta(p, u)$ and $\delta(q, u)$ are both defined and therefore the left-most letter of u must have the same distance marking as p and q.

If p and q are on different branches of the thick part of ${}^{\rm inv}\mathcal{B}_i^0$ or ${}^{\rm inv}\mathcal{B}_i^1$ (see Figures 1 and 2) then many cases are possible. The case where p and q are inside the diverging tree or the converging tree will also be handled by these cases.

Let us first consider the case of automata of the form ${}^{\rm inv}\mathcal{B}_i^0$.

Case 1 Suppose that along the path from p to q labeled by u, the thin part of ${}^{\rm inv}\mathcal{B}_i^0$ is visited. Then we can write $u = u_1u_2$ where u_1 is the shortest prefix of u such that $\delta(p, u_1) = p_1$ belongs to the thin part of ${}^{\rm inv}\mathcal{B}_i^0$; also, $\delta(p_1, u_2) = q$. Now $\delta(q, u_1u_2)$ and $\delta(p_1, u_2)$ are both defined, so $dist(\delta(q, u_1)) = dist(p_1)$ (the distance marking of the left-most letter of u_2). But the distance markings on the states in the thin part of ${}^{\rm inv}\mathcal{B}_i^0$ are different from the distance markings of the states in the thick part, and in the thin part each distance mark occurs on only one state (except for the final state, at distance 2, but that state is never visited before the end of the reading of a reduced word). So $\delta(q, u_1) = p_1 = \delta(p, u_1)$, and hence $q = p$ by injectivity.

Case 2 Suppose that neither p nor q belong to the thin part of ${}^{\rm inv}\mathcal{B}_i^0$, and that the thin part is not visited along the path from p to q labeled by u. Then the path goes through the converging tree or the diverging (see Figure 2) — unless p and q are on the same branch of the thick part of ${}^{\text{inv}}\mathcal{B}_i^0$, in which case $p = q$ (since on any branch there is only one state with a given distance). Let us now consider all the cases depending on the different branches that p and q can be in.

Since we are now assuming that the path from p to q, labeled by u, stays within the thick part of ${}^{\text{inv}}\mathcal{B}_i^0$, the path does not visit the root of the diverging or the converging tree (otherwise we already proved that $p = q$).

Within the diverging or the converging tree, we say that the root has *depth* 0, the vertices directly connected to the root have depth 1, etc.

Case 2.1 p is in a branch $(q_{1_r}, a_1, *)$ — $\left(*, \binom{a_1}{p_1^r}\right)$ and q is in a branch $(q_{2_r}, a_2, *)$ — $\left(*, \binom{a_2}{p_2^r}\right)$. See Figure 3.

If the path from p to q (labeled by u) goes through vertex 2_{conv} (the vertex labeled "2" in the converging tree), then $p = q$. Indeed, we can follow the path labeled by u, starting at p, and the path labeled by u, starting at q. These two paths are "synchronous," in the sense that the distance markings of u force the distances of the vertices reached in one path or the other to be the same at any moment. Thus, when the path from p reaches 2_{conv} for the first time, the path from q must also be at a vertex at depth 1 in the converging tree; the only vertices at depth 1 in the converging tree are 2_{conv} and the vertices labeled $p^r \in Q^r$; but the latter are not reachable from the branches that p and q are in (unless one visits the root of the converging tree or of the diverging tree). We conclude that there is a prefix v of u such that $\delta(q, v) = 2_{\text{conv}} = \delta(p, v)$. By injectiveness of ${}^{\text{inv}}\mathcal{B}_i$ we then obtain $p = q$.

If the path from p to q does not visit 2_{conv}, then we must have $q_{1_r} = q_{2_r} (= q_r)$ and the path visits the vertex q_r in the diverging tree (unless $a_1 = a_2$ as well, but then p and q are in the same branch, and that case was handled already at the beginning of case (2)). Again, we follow the two paths (one starting from p, the other starting from q), labeled by u; they are synchronous. After some prefix v of u was read, the path from p reaches q_r in the diverging tree, and the path from q also reaches a vertex at depth 1 in the diverging tree; but the only vertex the path from q could then be at is q_r (since all other vertices at depth 1 are unreachable from q, if 2_{conv} and the roots of the trees are not visited). Thus $\delta(q, v) = q_r = \delta(p, v)$, so $p = q$ by injectiveness.

Case 2.2 p is in a branch $(q_r, a, *)$ — $\left(*, \binom{a}{p^r}\right)$, and q is not in a branch of that form.

In this case, the path from p to q must go through vertex 2_{conv}, since the roots cannot be visited (see Figure 3). By the same reasoning as in the first part of case 2.1, we obtain a prefix v of u such that $\delta(p, v) = 2_{\text{conv}}$ and $\delta(q, v)$ is at depth 1 in the converging tree, hence $\delta(q', v)$ is either 2_{conv} or a vertex in the converging tree labeled p_1^r (for some $p_1^r \in Q^r$). We can assume that v is the shortest prefix of u with the above property. Then the last (rightmost) letter of v must be $\binom{a}{p^r}$. Since no edge with label $\binom{a}{p^r}$ points to a vertex p_1^r in the converging tree, we can rule out that $\delta(q, v) = p_1^r$. Thus $\delta(q, v) = 2_{\text{conv}} = \delta(p, v)$. By injectiveness of the automaton we conclude that $p = q$.

Case 2.3 p is in a branch $\left(\binom{a_1}{q_{1_\ell}}, *\right)$ — $(*, p_1^\ell, a_1)$ and q is in one of the branches $\left(\binom{a_2}{q_{2_\ell}}, *\right)$ — $(*, p_2^\ell, a_2)$, or $(a_2, *)$ — $(*, a_2)$, or $(a_2, q_{2_\ell}, *)$ — $\left(*, \binom{a_2}{p_2^\ell}\right)$.

The path from p to q visits either 2_{div} or 2_{conv} or $(a_1)_{\text{conv}}$ (the vertex in the converging tree labeled by $a_1 \in \Gamma$).

(1) If 2_{div} is visited, let v be the shortest prefix of u such that $\delta(p, v) = 2_{\text{div}}$; the last (i.e., rightmost) two letters of v must belong to $\Gamma^{-1}\binom{a}{q_\ell}^{-1} \cup \Gamma^{-1}\binom{a}{q_w}^{-1} \cup (\Gamma \cup \Gamma_{Q_r} \cup \Gamma_{Q_w})^{-1} a^{-1}$ for some $a \in \Gamma$, $q_\ell \in Q_\ell$, $q_w \in Q_w$. By the synchronousness argument (see Case 2.1), $\delta(q, v)$ must be either 2_{div}, or a vertex of the diverging tree labeled q_r for some $q_r \in Q_r$. But since the last two letters of v belong to the set above, the latter is ruled out. Thus $\delta(q, v) = 2_{\text{div}} = \delta(p, v)$, hence $p = q$ by injectiveness.

(2) If 2_{div} is not visited but 2_{conv} is visited, then let v be the shortest prefix of u such that $\delta(p, v) = 2_{\text{conv}}$. By the synchronousness argument, $\delta(q, v)$ is either 2_{conv} or $(p^r)_{\text{conv}}$ for some $p^r \in Q^r$; but since 2_{div} and the roots are not visited, $(p^r)_{\text{conv}}$ is not reachable from the branch p is in. Thus $\delta(q, v) = 2_{\text{conv}} = \delta(p, v)$, hence $p = q$ by injectiveness.

(3) Finally, there is the possibility that neither 2_{div} nor 2_{conv} are visited. Then we must have $a_1 = a_2$ (say, $= a$) and the vertex a_{conv} is visited. Let v be the shortest prefix of u such that $\delta(p, v) = a_{\text{conv}}$. By synchronousness, $\delta(q, v)$ is also a vertex of depth 2 in the converging tree. Since 2_{div} and 2_{conv} are not visited, the only vertex at depth 2 that can be visited on the path from p to q is a_{conv}. Thus $\delta(p, v) = a_{\text{conv}} = \delta(q, v)$, so $p = q$.

There are many more cases regarding the branches that contain p and q. All these cases are very similar to the ones we treated in detail above.

We have considered the cases where the path from p to q is entirely in the "thick" part of ${}^{\text{inv}}\mathcal{B}_i^0$. To finish off ${}^{\text{inv}}\mathcal{B}_i^0$ we still have to consider the case where $p = f$ or $q = f$.

If $p = f$ then (since $\delta(p, u)$ is defined) the left-most (marked) letter of u must be $(\#, 1)^{-1}$ (see Figure 2). But the only state of ${}^{\text{inv}}\mathcal{B}_i^0$ on which this letter is defined is f; thus (since $\delta(q, u)$ is defined), we must have $q = f$. Therefore $q = p$.

If $q = f$ we prove in exactly the same way that then $p = f$. Thus again $q = p$.

Let us finally consider ${}^{\text{inv}}\mathcal{B}_i^1$.

It only differs from the case of ${}^{\text{inv}}\mathcal{B}_i^0$ by the presence of the $s - s_1$ branch and the $t_1 - f$ branch (see Figure 1), so we only have to handle the cases where p or q belongs to one of these branches (all other cases were handled when we studied ${}^{\text{inv}}\mathcal{B}_i^0$).

If p and q both belong to the $s - s_1$ branch (or both belong to the $t_1 - f$ branch) then $p = q$ since $dist(p) = dist(q)$. So, for the rest of the proof we assume that p and q do not belong to the same such branch.

Claim Suppose p belongs to the $s - s_1$ branch of ${}^{\text{inv}}\mathcal{B}_i^1$ and $\delta(p, u^m) = p$ and $\delta(p, u) = q$. Then $\delta(s_1, z^{-1}uz) \neq s_1$ and $\delta(s_1, (z^{-1}uz)^m) = s_1$, where z is the label of the shortest path from p to s_1 (see Figure 1).

Proof of the claim. Since q does not belong to the $s - s_1$ branch, the path from p to q labeled by u must visit s_1. Since we assume that u is group-reduced, we conclude from $\delta(p, u^m) = p$ that z is a prefix of u and that z^{-1} is a suffix of u. Thus $\delta(s_1, z^{-1}\, uz)$ and $\delta(s_1, (z^{-1}uz)^m)$ are defined in $\mathcal{B}$. Clearly $\delta(s_1, (z^{-1}uz)^m) = s_1$; moreover, $\delta(s_1, z^{-1}uz) \neq s_1$, otherwise $\delta(p, u)$ would be equal to p. □

The above claim reduces the problem to the states s_1 and $\delta(s_1, z^{-1}uz)$, which correspond to cases that were considered when we studied ${}^{\text{inv}}\mathcal{B}_i^0$. That is, we already know that $s_1 = \delta(s_1, z^{-1}uz)$, and hence $p = q$. This contradicts the hypothesis that p and q do not belong to the same branch.

On the other hand, if q belongs to the $s - s_1$ branch of ${}^{\text{inv}}\mathcal{B}_i^1$, then u^{-1} is a reduced word such that $\delta(q, u^{-1}) = p$ and $\delta(q, u^{-m}) = \delta(p, u^{-m+1}) = \delta(p, u^{2m+1}) = \delta(p, u) = q$. So the above reasoning shows that, here again, the hypothesis that p and q do not belong to the same branch, leads to a contradiction.

The cases where p or q is in the $t_1 - f$ branch of ${}^{\text{inv}}\mathcal{B}_i^1$ are handled in the same way. This completes the proof that every ${}^{\text{inv}}\mathcal{B}_i^0$, ${}^{\text{inv}}\mathcal{B}_i^1$ is aperiodic. □

Lemma 6.6 *If the Turing machine $\mathcal{T}$ accepts w, then the automaton ${}^{\text{inv}}\mathcal{B} = {}^{\text{inv}}\mathcal{B}(w)$ is not aperiodic.*

Proof. Let C be the word $\#c_0\#c_1\cdots\#c_k\#\#$ corresponding to the accepting computation, rewritten over the new alphabet $\Delta_{\mathcal{B}} \cup \Delta_{\mathcal{B}}^{-1}$ with distance marks. Then C is accepted by each $\mathcal{B}_i^0$ and $\mathcal{B}_i^1$, and hence it is also accepted by each ${}^{\mathrm{inv}}\mathcal{B}_i^0$ and ${}^{\mathrm{inv}}\mathcal{B}_i^1$. As a consequence $\delta(s^{(0)}, (C\flat)^P) = s^{(0)}$ in ${}^{\mathrm{inv}}\mathcal{B}$; but we also have $\delta(s^{(0)}, C\flat) = s^{(1)} \neq s^{(0)}$. Thus ${}^{\mathrm{inv}}\mathcal{B}$ is not aperiodic. $\square$

The proof of the converse of Lemma 6.6 is more difficult and is given in the next lemmas. For this proof it will be convenient to rename the component automata of ${}^{\mathrm{inv}}\mathcal{B}$. We let

$$\begin{aligned} \mathcal{B}_h^0 &= \mathcal{B}(2h) && \text{for } 1 \leq h \leq (P-1)/2, \text{ and} \\ \mathcal{B}_h^1 &= \mathcal{B}(2h+1) && \text{for } 1 \leq h \leq (P+1)/2. \end{aligned}$$

So in this notation the component automata are $\mathcal{B}(k)$, $k = 1, \ldots, P$ (in cyclic order). As a structure we denote $\mathcal{B}(k) = (Q_k, \Delta_{\mathcal{B}}, \delta_k, s^{(k)}, \{f^{(k)}\})$.

Suppose that ${}^{\mathrm{inv}}\mathcal{B}$ is not aperiodic. Then there exist a state p of ${}^{\mathrm{inv}}\mathcal{B}$, a word $u \in (\Delta_{\mathcal{B}} \cup \Delta_{\mathcal{B}}^{-1} \cup \{\flat, \flat^{-1}\})^*$ and an integer $m > 1$ such that $\delta(p, u^m) = p$ and $\delta(p, u) \neq p$. Let m be the minimum integer such that there exist such a state p and such a word u. Observe that m is necessarily prime. Otherwise, m can be factored as $m = m_1 m_2$ with $m_1, m_2 < m$. Then letting $v = u^{m_1}$, we have $\delta(p, v^{m_2}) = p$. Since m is minimum, $\delta(p, u^h) \neq p$ for all $h < m$, so $\delta(p, v) = \delta(p, u^{m_1}) \neq p$. But this contradicts the minimality of m since $m_2 < m$.

For this m, we choose p and u so that u has minimum length. In particular, u is group-reduced (by Lemma 1.1).

Let $q = \delta(p, u)$. We want to show that then

$$\bigcap_{h=1}^{P} {}^{\mathrm{inv}}\mathcal{B}(h) \;=\; \bigcap_{i=1}^{(P-1)/2} L({}^{\mathrm{inv}}\mathcal{B}_i^0) \;\cap\; \bigcap_{i=1}^{(P+1)/2} L({}^{\mathrm{inv}}\mathcal{B}_i^1) \;\neq\; \emptyset.$$

This (by Section 5.2) means that w is accepted by $\mathcal{T}$.

Our proof will again start out like the proof in [7], but then it has to handle the cycles created by the inverse edges in ${}^{\mathrm{inv}}\mathcal{B}$.

Suppose p is a state of the component automaton ${}^{\mathrm{inv}}\mathcal{B}(i)$, and q is a state of the component automaton ${}^{\mathrm{inv}}\mathcal{B}(j)$ $(1 \leq i, j \leq P)$. Recall that $p \neq q$.

Lemma 6.7 *The distance of p in its component automaton ${}^{\mathrm{inv}}\mathcal{B}(i)$ is equal to the distance of q in its component automaton ${}^{\mathrm{inv}}\mathcal{B}(j)$.*

Proof. Since $\delta(p, u)$ and $\delta(q, u)$ are both defined, the marking on the left-most letter of u must be equal to both $dist(p)$ and $dist(q)$. $\square$

Lemma 6.8 *The states p and q belong to different component automata. That is, $i \neq j$.*

Proof. Suppose, by contradiction, that $i = j$; so both p and q are states of ${}^{\mathrm{inv}}\mathcal{B}(i)$.

The path in ${}^{\mathrm{inv}}\mathcal{B}$ from p to q, labeled by u, must *exit* ${}^{\mathrm{inv}}\mathcal{B}(i)$. Otherwise u does not contain the letter $\flat$ nor its inverse $\flat^{-1}$. But then the path starting and ending at p, labeled by u^m, does not exit ${}^{\mathrm{inv}}\mathcal{B}(i)$ either, which implies that ${}^{\mathrm{inv}}\mathcal{B}(i)$ is not aperiodic, in contradiction with Lemma 6.5.

The path from p to q, labeled by u, can *reenter* into ${}^{\mathrm{inv}}\mathcal{B}(i)$ either at the start state $s^{(i)}$ or at the accept state $f^{(i)}$ (here we do not care where along the path we exit from ${}^{\mathrm{inv}}\mathcal{B}(i)$). In fact, it is the occurrences of $\flat$ and $\flat^{-1}$ in u that determine the sequence of component automata ${}^{\mathrm{inv}}\mathcal{B}(k)$ visited along the path from p to q labeled by u. Since we start and end in ${}^{\mathrm{inv}}\mathcal{B}(i)$, the number of occurrences of $\flat$ in u is equal, modulo P, to the number of occurrences of $\flat^{-1}$ in u. It follows that on the path from p to p labeled by u^m, the same pattern of visits of automata ${}^{\mathrm{inv}}\mathcal{B}(k)$ happens as for u, but repeated m times.

(1) Let us first consider the case where i is even, that is ${}^{\mathrm{inv}}\mathcal{B}(i)$ is of the form ${}^{\mathrm{inv}}\mathcal{B}_h^0$, and where the path reenters through state $s^{(i)}$. Then u can be written as $u = u_1u_2$ where u_2 is the shortest suffix of u such that $\delta(p, u_1) = s^{(i)}$ and $\delta(s^{(i)}, u_2) = q$. By the above remarks, the path from p to p labeled by u^m also reenters ${}^{\mathrm{inv}}\mathcal{B}(i)$ through s. So, we can also write u^m as $u^m = v_1v_2$ where v_2 is the shortest suffix of u^m such that $\delta(p, v_1) = s^{(i)}$ and $\delta(s^{(i)}, v_2) = p$. Since both u_2 and v_2 are suffixes of u^m, either u_2 is a suffix of v_2 or v_2 is a suffix of u_2. So there exists a string y such that either $v_2 = yu_2$ or $u_2 = yv_2$. In either case, we have $\delta(s^{(i)}, y) = s^{(i)}$. Indeed, the distance markings on u_2 and v_2 are such that u_2 and v_2 are only defined on $s^{(i)}$, and on no other state ($s^{(i)}$ is the only state of ${}^{\mathrm{inv}}\mathcal{B}_h^0$ with $dist = 0$, see Figure 2). Since u_2 and v_2 were chosen to be of minimum length, y must therefore be the empty string. Thus $u_2 = v_2$. But then $q = \delta(s^{(i)}, u_2) = \delta(s^{(i)}, v_2) = p$. So, $p = q$, contradicting our hypothesis that $p \neq q$.

(2) Let us consider next the case where the path from p to q labeled by u reenters ${}^{\mathrm{inv}}\mathcal{B}(i) = {}^{\mathrm{inv}}\mathcal{B}_h^0$ through state $f^{(i)}$. Now we do the same reasoning as in case (1), but we use the state t_1 of ${}^{\mathrm{inv}}\mathcal{B}_h^0$ (see Figure 2) instead of $s^{(i)}$ (noting that t_1 is the only state of ${}^{\mathrm{inv}}\mathcal{B}_h^0$ with $dist = 1$). After the path from p to q labeled by u reenters through $f^{(i)}$, it must also pass through t_1 unless $q = f$; but we cannot have $q = f$ by the following argument:

If we had $q = f$ then the leftmost letter of u would have to be either $\flat$ or $(\#, 1)^{-1}$, since $\delta(q, u)$ is defined. Moreover, the only state on which any one of these two letters is defined is f. Thus $p = f$, since $\delta(p, u)$ is defined. But now $p = f = q$, which contradicts the fact that $p \neq q$.

(3) Let us now consider the case where i is odd, that is ${}^{\mathrm{inv}}\mathcal{B}(i)$ is of the form ${}^{\mathrm{inv}}\mathcal{B}_h^1$, and suppose that the path from p to q labeled by u reenters into ${}^{\mathrm{inv}}\mathcal{B}(i)$ through $s^{(i)}$.

If this path visits state s_1 (which is the only state of ${}^{\mathrm{inv}}\mathcal{B}(i) = {}^{\mathrm{inv}}\mathcal{B}_h^1$ with $dist = n+1$, see Figure 1) then the reasoning of case (1) can be repeated, with $s^{(i)}$ replaced by s_1.

If this path never visits s_1, but visits state t_1 (which is the only state of ${}^{\mathrm{inv}}\mathcal{B}(i) =^{\mathrm{inv}} \mathcal{B}_h^1$ with $dist = n+2$) then again, the reasoning of case (1) can be used, with $s^{(i)}$ replaced by t_1.

If neither s_1 nor t_1 are visited on the path from p to q labeled by u (while still assuming that the path exits from ${}^{\mathrm{inv}}\mathcal{B}(i)$, and reenters through $s^{(i)}$) then either p and q are both on the $s - s_1$ branch or they are both on the $t_1 - f$ branch of ${}^{\mathrm{inv}}\mathcal{B}(i) = {}^{\mathrm{inv}}\mathcal{B}_h^1$. But then, the fact that $dist(p) = dist(q)$ implies $p = q$, which contradicts the fact that $p \neq q$.

(4) Finally, we consider the case where i is odd and the path from p to q reenters into ${}^{\mathrm{inv}}\mathcal{B}(i)$ through $f^{(i)}$. This is entirely like case (3). $\square$

Now, let i and j be the indices of the component automata of ${}^{\mathrm{inv}}\mathcal{B}$ containing p and q respectively. That is, p is a state of ${}^{\mathrm{inv}}\mathcal{B}(i)$ and q is a state of ${}^{\mathrm{inv}}\mathcal{B}(j)$. By the previous lemma, $i \neq j$. Let D be the integer satisfying $0 < D < P$ and $D \equiv (j - i) \bmod P$.

Let also $p_0 = p$ and for any $k > 0$, let $p_k = \delta(p, u^k)$. Then in particular, $p_1 = q$ and $p_m = p$. For every $k \geq 0$, let $i(k)$ be the index of the component automaton of ${}^{\mathrm{inv}}\mathcal{B}$ that p_k is a state of. Then $i(0) = i$ (since $p = p_0$ is in ${}^{\mathrm{inv}}B(i)$), $i(1) = j$ (since $q = p_1$ is in ${}^{\mathrm{inv}}B(j)$), and $i(m) = i(0)$.

Just as in [7] we have the following lemma.

Lemma 6.9 *With the above notation, we have*

(a) $i(k+1) - i(k) \equiv D \bmod P$, *for all* $k \geq 0$.

(b) *$m = P$, and $\{i(k) \mid 1 \le k \le P\} = \{i \mid 1 \le i \le P\}$.*

Proof. (a) As remarked in the proof of the previous lemma (see also Figure 4), it is the pattern of occurrences of the letter $\flat$ and its inverse $\flat^{-1}$ in u that determines the difference $i(k+1)-i(k)$. Exactly, we have

$$i(k+1) - i(k) \equiv |u|_\flat - |u|_{\flat^{-1}} \bmod P$$

where $|u|_a$ denotes the number of occurrences of letter a in u. This number does not depend on k.
(b) Since $i(m) = i(0)$, part (a) implies that $mD \equiv 0 \bmod P$, that is, P divides mD. Now $0 < D < P$ and both P and m are prime (recall that m is prime by minimality). It follows that $m = P$. This in turn implies

$$\{i(k) \mid 1 \le k \le m\} = \{kD \mid 1 \le k \le m\} = \{i \mid 1 \le i \le P\}$$

again using the primality of $m = P$. □

Lemma 6.10 *The word u can be factored as $u = u_1u_2u_3$ in such a way that*

- *for each $0 \le k < m$, the path labeled u_1 from p_k to $\delta(p_k, u_1)$ is entirely contained in ${}^{\mathrm{inv}}\mathcal{B}(i(k))$;*
- *for each $0 \le k < m$, the path labeled u_2 from $\delta(p_k, u_1)$ to $\delta(p_k, u_1u_2)$ does not visit ${}^{\mathrm{inv}}\mathcal{B}(i(k))$ nor ${}^{\mathrm{inv}}\mathcal{B}(i(k+1))$ (except at the beginning and the end);*
- *for each $0 \le k < m$, the path labeled u_3 from $\delta(p_k, u_1u_2)$ to p_{k+1} is entirely contained in ${}^{\mathrm{inv}}\mathcal{B}(i(k+1))$.*

Moreover, either $\delta(p_k, u_1) = f^{(i(k))}$ and $\delta(p_k, u_1u_2) = s^{(i(k+1))}$ for all $0 \le k < m$, or $\delta(p_k, u_1) = s^{(i(k))}$ and $\delta(p_k, u_1u_2) = f^{(i(k+1))}$ for all $0 \le k < m$.

Proof. The path from p_k to p_{k+1} labeled u must contain some occurrence of $\flat$ or $\flat^{-1}$ since $D \ne 0 \bmod P$. Let u_1 be the longest prefix of u without occurrence of $\flat$ or $\flat^{-1}$ and let u_3 be the longest suffix of u without occurrence of $\flat$ or $\flat^{-1}$. Finally, let u_2 be such that $u = u_1u_2u_3$. Then $\delta(p_k, u_1)$ is the last state of ${}^{\mathrm{inv}}\mathcal{B}(i(k))$ visited before the first exit out of that component automaton along the path labeled u from p_k to p_{k+1}. Similarly, $\delta(p_k, u_1u_2)$ is the last entry into ${}^{\mathrm{inv}}\mathcal{B}(i(k+1))$ along that path.

Since $i(k+1) - i(k) \equiv D \bmod P$, we have $|u_2|_\flat - |u_2|_{\flat^{-1}} \equiv D \bmod P$. Let v_1 be the shortest prefix of u_2 such that $|v_1|_\flat - |v_1|_{\flat^{-1}} \equiv D \bmod P$, and let v_2 be such that $u_2 = v_1v_2$. Then $\delta(p_k, u_1v_1)$ is the first entry into ${}^{\mathrm{inv}}\mathcal{B}(i(k+1))$ along the path labeled u from p_k to p_{k+1}.

Now the only entry points into a component automaton ${}^{\mathrm{inv}}\mathcal{B}(h)$ are $s^{(h)}$ (after reading a $\flat$) and $f^{(h)}$ (after reading a $\flat^{-1}$). Similarly, the exit out of ${}^{\mathrm{inv}}\mathcal{B}(h)$ is through $s^{(h)}$ (before reading a $\flat^{-1}$) or $f^{(h)}$ (before reading a $\flat$). In other words, we have either $\delta(p_k, u_1) = s^{(i(k))}$ for all k, or $\delta(p_k, u_1) = f^{(i(k))}$ for all k. Similar statements hold for $\delta(p_k, u_1v_1)$ and $\delta(p_k, u_1u_2)$.

First we show that $v_2 = 1$. If $\delta(p_k, u_1v_1) = \delta(p_k, u_1u_2)$ (as we just saw, if it happens for some k, then it happens for all k), then we have $\delta(p_k, u_1v_1u_3) = p_{k+1}$ for each k, and this contradicts the minimality of $|u|$.

Let us now assume that $\delta(p_k, u_1v_1) \ne \delta(p_k, u_1u_2)$, say $\delta(p_k, u_1v_1) = s^{(i(k+1))}$ and $\delta(p_k, u_1u_2) = f^{(i(k+1))}$. By definition of v_2, the path from $s^{(i(k+1))}$ to $f^{(i(k+1))}$ labeled by v_2 exits ${}^{\mathrm{inv}}\mathcal{B}(i(k+1))$, that is, v_2 contains occurrences of $\flat$ or $\flat^{-1}$. Let v_3 be the longest prefix of v_2 containing no occurrence of $\flat$ or $\flat^{-1}$. Since u is reduced and $\delta(p_k, u_1v_1)$ and $\delta(p_k, u_1v_1v_2) = \delta(p_k, u_1u_2)$ are entry points into ${}^{\mathrm{inv}}\mathcal{B}(i(k+1))$, we have $v_2 = v_3v_4$ with $v_3, v_4 \neq 1$. Moreover $\delta(p_k, u_1v_1v_3) \in \{s^{(i(k+1))}, f^{(i(k+1))}\}$. So either v_3 labels a loop around $s^{(i(k+1)}$ for all k, or v_4 labels a loop around $f^{(i(k+1)}$ for all k. In either case, we contradict the minimality of $|u|$ since either $\delta(p_k, u_1v_1v_4u_3) = p_{k+1}$ for all k, or $\delta(p_k, u_1v_1v_3u_3) = p_{k+1}$ for all k. The situation is entirely symmetrical if $\delta(p_k, u_1v_1) = f^{(i(k+1))}$ and $\delta(p_k, u_1u_2) = s^{(i(k+1))}$.

So we have shown that $v_2 = 1$, that is, the path labeled u from p_k to p_{k+1} enters ${}^{\mathrm{inv}}\mathcal{B}(i(k+1))$ exactly once, namely after having read the prefix $u_1u_2 = u_1v_1$ of u.

A similar reasoning involving the longest prefix w_1 of u_2 such that $|w_1|_\flat - |w_1|_{\flat^{-1}} \equiv 0 \bmod P$ (that is, the last exit out of ${}^{\mathrm{inv}}\mathcal{B}(i(k))$) shows that the path labeled u from p_k to p_{k+1} exits ${}^{\mathrm{inv}}\mathcal{B}(i(k))$ exactly once, namely after having read the prefix u_1 of u.

Finally, let us observe that

$$\begin{array}{lll} \text{if } \delta(p_k, u_1) = s^{(i(k))} & \text{then} & \delta(p_k, u_1u_2) = f^{(i(k+1))}, \text{ and} \\ \text{if } \delta(p_k, u_1) = f^{(i(k))} & \text{then} & \delta(p_k, u_1u_2) = s^{(i(k+1))}. \end{array}$$

Indeed, a path from $s^{(i(k))}$ to $s^{(i(k+1))}$ (resp. $f^{(i(k))}$ to $f^{(i(k+1))}$) must travel throughout ${}^{\mathrm{inv}}\mathcal{B}(i(k))$ or ${}^{\mathrm{inv}}\mathcal{B}(i(k+1))$ (see Figure 4). This completes the proof of the lemma. □

Corollary 6.11 *Either u_3u_1 labels a path from $s^{(i(k))}$ to $f^{(i(k))}$ in each automaton ${}^{\mathrm{inv}}\mathcal{B}(i(k))$, or $u_1^{-1}u_3^{-1}$ labels a path from $s^{(i(k))}$ to $f^{(i(k))}$ in each automaton ${}^{\mathrm{inv}}\mathcal{B}(i(k))$.*

Proof. This follows immediately from Lemma 6.10. □

We can now complete the proof of the converse of Lemma 6.6.

Lemma 6.12 *If ${}^{\mathrm{inv}}\mathcal{B}$ is not aperiodic, then w is accepted by $\mathcal{T}$.*

Proof. Indeed, under the hypothesis that there exists a word u, an integer m and distinct states p and q such that $\delta(p, u) = q$ and $\delta(p, u^m) = p$, we have constructed in Corollary 6.11 a word which is accepted by each ${}^{\mathrm{inv}}\mathcal{B}(i(k))$ $(1 \leq k \leq P)$. We also know that $\{i(k) \mid 1 \leq k \leq P\} = \{i \mid 1 \leq i \leq P\}$ (Lemma 6.9). Hence

$$\bigcap_{1\leq i\leq P} L({}^{\mathrm{inv}}\mathcal{B}(i)) = \bigcap_{1\leq i\leq n} (L({}^{\mathrm{inv}}\mathcal{B}_i^0) \cap L({}^{\mathrm{inv}}\mathcal{B}_i^1)) \neq \emptyset.$$

This implies (by dropping the distance markings on the letters) that

$$\bigcap_{1\leq i\leq n} (L({}^{\mathrm{inv}}\mathcal{A}_i^0) \cap L({}^{\mathrm{inv}}\mathcal{A}_i^1)) \neq \emptyset.$$

In Section 5 we proved that the latter is equivalent to the fact that w is accepted by the Turing machine $\mathcal{T}$. □

This completes the proof of Proposition 6.4.

6.3 The fixed-alphabet aperiodicity problem for inverse finite automata

The alphabet $\Delta_{\mathcal{B}} \cup \Delta_{\mathcal{B}}^{-1}$ of the inverse finite automaton ${}^{\text{inv}}\mathcal{B}(w)$ (whose aperiodicity is equivalent to the non-acceptance of w by the Turing machine $\mathcal{T}$) has variable size (depending on the length of w). Thus our work does not immediately imply the PSPACE-completeness of the aperiodicity problem for inverse automata. We now prove the following theorem.

Theorem 6.13 *For each large enough alphabet, the fixed-alphabet aperiodicity problem for inverse finite automata is* PSPACE*-complete.*

In order to establish this result, we need to "encode" our construction in a fixed alphabet. This alphabet will be $\Delta' = \{0, 1, \#, \flat\} \cup \Delta$ (where Δ was defined in the description of the Turing machine $\mathcal{T}$); we also use inverses for all these letters.

The only reason why $\Delta_{\mathcal{B}}$ depends on w is because we marked letters by distances. So, to obtain a fixed alphabet, independent of w, we just have to encode the distance markings.

For every distance d ($0 \leq d \leq 2n+1$), let $\beta(d) \in \{0,1\}^+$ be the binary representation of d. Then the elements of $\Delta_{\mathcal{B}}$ are encoded by words of Δ'^* as follows:

$$\begin{aligned} \kappa(a, d) &= a\beta(d)a \\ \kappa(\flat) &= \flat. \end{aligned}$$

For each element x of $\Delta_{\mathcal{B}}$, we define $\kappa(x^{-1})$ to be $\kappa(x)^{-1}$, i.e., the formal inverse of the string $\kappa(x)$. The words of the form $\kappa(x)$ with $x \in \Delta_{\mathcal{B}} \cup {\Delta_{\mathcal{B}}}^{-1}$ are called *code words*. Observe that the set of code words forms a biprefix code: no code word is a prefix or a suffix of another code word. The maximal length of a code word is $2 + \lceil \log_2(2n+1) \rceil$. We extend κ to a homomorphism from $(\Delta_{\mathcal{B}} \cup {\Delta_{\mathcal{B}}}^{-1})^*$ to $(\Delta' \cup {\Delta'}^{-1})^*$.

By using this code we also encode the automaton ${}^{\text{inv}}\mathcal{B}$, in order to obtain a finite automaton $\kappa({}^{\text{inv}}\mathcal{B})$, which accepts $\kappa(L({}^{\text{inv}}\mathcal{B}))$. This is done by replacing every edge $p \xrightarrow{x} q$ of ${}^{\text{inv}}\mathcal{B}$ (where $x \in \Delta' \setminus \{\flat\}$) by a branch (of $|\kappa(x)|$ edges) labeled by $\kappa(x)$; $|\kappa(x)| - 1$ new states are introduced. We also introduce the inverses of the new edges, thus obtaining a path from q to p labeled by $\kappa(x^{-1})$. No replacement is performed on the $\flat$-edges and their inverses. We denote the transition function of $\kappa({}^{\text{inv}}\mathcal{B})$ by δ'.

Observe that $\kappa({}^{\text{inv}}\mathcal{B})$ is again an inverse automaton. Indeed each automaton $\mathcal{A}_i^0$ and $\mathcal{A}_i^1$ (the unmarked versions of the $\mathcal{B}(i)$) is inverse, and the encoding of the letters of the form (a, d) ($a \in \Delta \cup \{\#\}$) starts and ends with a.

Clearly if ${}^{\text{inv}}\mathcal{B}$ has a cycle (i.e., there exist p, u, m such that $p = \delta(p, u^m) \neq \delta(p, u)$) then $\kappa({}^{\text{inv}}\mathcal{B})$ also has a cycle (more precisely, $p = \delta'(p, \kappa(u)^m) \neq \delta'(p, \kappa(u))$). In order to prove the converse (namely, if ${}^{\text{inv}}\mathcal{B}$ is aperiodic then $\kappa({}^{\text{inv}}\mathcal{B})$ is also aperiodic) we will first prove the following.

Recall that a reduced word w is cyclically reduced if all its powers are reduced.

Lemma 6.14 *Let p be a state of $\kappa({}^{\text{inv}}\mathcal{B})$, let $m > 1$, and let u be a cyclically reduced word in $(\Delta' \cup {\Delta'}^{-1})^*$ such that in $\kappa({}^{\text{inv}}\mathcal{B})$, $\delta'(p, u) \neq p$ and $\delta'(p, u^m) = p$.*

Then u can be factored as $u = xyz$ such that y and zx are (possibly empty) products of code words, and the states $\delta'(p, u^h x)$ ($h \geq 0$) are states that were already in ${}^{\text{inv}}\mathcal{B}$.

Proof. The path from p to $\delta'(p, u)$ labeled u cannot be entirely contained within one of the branches added in the construction of $\kappa({}^{\text{inv}}\mathcal{B})$. Indeed we would then have $u \in \{0,1\}^+ \cup \{0^{-1}, 1^{-1}\}^+$, by definition of κ and because u is reduced. But for such a word u, $\delta'(p, u^k)$ is undefined for $k \geq 2 + \lceil \log_2(2n+1) \rceil$, a contradiction.

Let x be the shortest prefix of u (possibly the empty word) such that $\delta'(p,x)$ is a state of ${}^{\mathrm{inv}}\mathcal{B}$. Let also z be the shortest suffix of u (possibly the empty word) such that $\delta'(p,uz^{-1})$ is a state of ${}^{\mathrm{inv}}\mathcal{B}$. Then $u = xyz$, and y is a (possibly empty) reduced word which labels a path in $\kappa({}^{\mathrm{inv}}\mathcal{B})$ between two states that were already in ${}^{\mathrm{inv}}\mathcal{B}$, so y is a product of code words. Note that x (resp. z) is a proper suffix (resp. prefix) of a code word.

Similarly, u^2 can be factored as $u^2 = xyzxyz = x'y'z'$ where y' is a product of code words and x' (resp. z') is the shortest prefix (resp. suffix) of u^2 such that $\delta'(p,x')$ (resp. $\delta'(p,u^2z'^{-1})$) is a state of ${}^{\mathrm{inv}}\mathcal{B}$. Then evidently $x = x'$.

If z has length at least 2, then z ends with one of 0, 1, 0^{-1} and 1^{-1}, and hence so does z'. Therefore both z and z' are the shortest suffix of u^2 starting with a letter in $\Delta \cup \Delta^{-1} \cup \{\#, \#^{-1}\}$: so $z = z'$.

If z has length 1, then z is a letter in $\Delta \cup \Delta^{-1} \cup \{\#, \#^{-1}\}$, so z' ends with that letter. But z' is a proper prefix of a code word, so $|z'| = 1$, and hence $z = z'$.

If z' has length at least 1, the same reasoning holds, and we have $z = z'$. The last case is that where both z and z' have length zero, but this too implies $z = z'$.

From the equality $xyzxyz = x'y'z'$, it now follows that $yzxy = y'$. This implies that zx is a (possibly empty) product of code words since the code words form a biprefix code.

For each $h \geq 0$, $\delta'(p,u^hx)$ is defined, since $\delta'(p,u^m) = p$. Moreover $u^hx = x(yzx)^h$, and $\delta'(p,x)$ is a state of ${}^{\mathrm{inv}}\mathcal{B}$. It follows from the construction of $\kappa({}^{\mathrm{inv}}\mathcal{B})$ that the action of a product of code words on a state of ${}^{\mathrm{inv}}\mathcal{B}$ leads again to a state of ${}^{\mathrm{inv}}\mathcal{B}$. This concludes the proof. □

Lemma 6.15 *$\kappa({}^{\mathrm{inv}}\mathcal{B})$ is aperiodic if and only if ${}^{\mathrm{inv}}\mathcal{B}$ is aperiodic.*

Proof. We mentioned already (before the previous lemma) that if ${}^{\mathrm{inv}}\mathcal{B}$ is not aperiodic then $\kappa({}^{\mathrm{inv}}\mathcal{B})$ is not aperiodic.

Let us prove the converse. If $\kappa({}^{\mathrm{inv}}\mathcal{B})$ is not aperiodic, there exists a state p, a reduced word u and an integer $m > 0$ such that $\delta'(p,u) \neq p$ and $\delta'(p,u^m) = p$. The reduced word u can be factored uniquely as $u = vwv^{-1}$, where w is cyclically reduced. But if $q = \delta'(p,v)$, then $\delta'(q,w) \neq q$ and $\delta'(q,w^m) = q$. So we can assume that u is cyclically reduced.

We use the notation of Lemma 6.14. Let $p' = \delta'(p,x)$. Then it is immediately verified that $\delta'(p',yzx) \neq p'$ and $\delta'(p',(yzx)^m) = p'$. Moreover, the $\delta(p',(yzx)^h)$ are states of ${}^{\mathrm{inv}}\mathcal{B}$, and $yzx = \kappa(v)$, for some word $v \in (\Delta_{\mathcal{B}} \cup \Delta_{\mathcal{B}}^{-1})^*$. Therefore we have $\delta(p',v) \neq p'$ and $\delta(p',v^m) = p'$ in ${}^{\mathrm{inv}}\mathcal{B}$. This proves that if $\kappa({}^{\mathrm{inv}}\mathcal{B})$ is not aperiodic then ${}^{\mathrm{inv}}\mathcal{B}$ is not aperiodic. □

We already know that the aperiodicity problem belongs to PSPACE (Section 4). Now we have completed the proof that the aperiodicity problem for inverse finite automata, with a fixed, large enough alphabet, is PSPACE-complete (since the size of $\kappa({}^{\mathrm{inv}}\mathcal{B})$ is polynomial in the size of ${}^{\mathrm{inv}}\mathcal{B}$).

7 The purity problem, the p-purity problem, and the p-automaton problem

We can now conclude this paper by proving the announced results.

Theorem 7.1 *For any free group G of large enough rank, the purity problem of G is* PSPACE-*complete.*

Proof. This follows immediately by combining Theorems 3.6 and 6.13. □

Theorem 7.2 *For any free group G of large enough rank and any prime number p, the p-purity problem of G for that prime p is* PSPACE*-complete.*

Proof. This follows immediately by combining Theorem 3.6 and the next theorem. □

Theorem 7.3 *For any large enough alphabet and any prime number p, the p-automaton problem for that alphabet and that prime p is* PSPACE*-complete.*

Proof. In the previous sections, we fixed a Turing machine $\mathcal{T}$ recognizing a PSPACE-complete language, with properties summarized in Corollary 4.3, and we constructed an inverse finite automaton ${}^{\text{inv}}\mathcal{B} = {}^{\text{inv}}\mathcal{B}(w)$ which is aperiodic if and only if $\mathcal{T}$ does not accept w. We also showed that when ${}^{\text{inv}}\mathcal{B}(w)$ is not aperiodic then it contains a P-cycle, where $P = P(w)$ is the smallest prime $\geq 2|w|$. Indeed we showed in Lemma 6.9 that for some choice of u, p, if $\delta(p, u) = q \neq p$ and $\delta(p, u^m) = p$, then m is a multiple of P.

Let p be any fixed prime. Let us modify the Turing machine $\mathcal{T}$ in such a way that it does not accept any input w of length $\leq p$ (this can be done easily, without modifying any of the other properties of this Turing machine, stated in Corollaries 4.2 and 4.3). So now, if the Turing machine accepts w we must have $p < |w| < P(w)$.

In summary we have proved the following:

If the Turing machine $\mathcal{T}$ accepts w then ${}^{\text{inv}}\mathcal{B}(w)$ contains a $P(w)$-cycle, with $P(w) > p$; so ${}^{\text{inv}}\mathcal{B}(w)$ is not a p-automaton.

Conversely, if $\mathcal{T}$ does not accept w then ${}^{\text{inv}}\mathcal{B}(w)$ is aperiodic; so it is a p-automaton.

This shows that the p-automaton problem is PSPACE-hard.

We still must show that the p-automaton problem is in PSPACE for any fixed prime number. We use a standard argument (used also in [13] and [26]). Note that, by Savitch's Theorem [12], PSPACE is the same class, whether we use determinism or non-determinism. Let $\mathcal{A}$ be any finite automaton. We do not need to assume that $\mathcal{A}$ is inverse or injective. The automaton $\mathcal{A}$ is not a p-automaton if and only if there exists a word $u \in \Gamma^*$ (where Γ is the alphabet of $\mathcal{A}$) and a state $q \in Q$ (where Q is the state set of $\mathcal{A}$) such that $\delta(q, u) \neq q$, and $\delta(q, u^m) = q$, where $m > 1$ is a number not divisible by p. Let $k = |Q|$. Then $m \leq k^k$ (since the transition monoid of $\mathcal{A}$ has at most k^k elements), so m can be represented (in binary) using space $O(k \cdot \log k)$.

To check the above we use a non-deterministic Turing machine which guesses q and m and writes them down (in space $O(k^2)$). It is straightforward to check (in space $O(k \cdot \log k)$) that $m > 1$ is not divisible by p. Next it guesses u, one letter after another as follows:

1. Guess the first letter a_1 of u; compute (and write down) the function table of the function $s \in Q \mapsto \delta(s, a_1) \in Q$; this can be done in linear space.
2. After guessing the first i letters $a_1, \ldots, a_i$ of u, suppose the function table of the function $s \in Q \mapsto \delta(s, a_1 \ldots a_i)$ was written down (we do not assume that the string $a_1 \ldots a_i$ was written down). Now guess the next letter a_{i+1} of u, and compute the function table of the function $s \mapsto \delta(s,\, a_1 \ldots a_i a_{i+1}) = \delta(\delta(s,\, a_1 \ldots a_i),\, a_{i+1})$. Replace the old function table by the new one. This takes linear space.
3. Guess that u is finished.

Now the function table of the function $\delta(\cdot, u)\colon s \mapsto \delta(s, u)$ is on a tape of the Turing machine. If $\delta(q, u) = q$, the Turing machine rejects (for this guessed word u and this guessed state q).

Finally, using the function $\delta(\cdot, u)$ and m, compute $\delta(q, u^m)$ by applying the function m times. This does not use any new space (in addition to the space $O(k^2)$ allocated to m and to the function table). If $\delta(q, u^m) = q$ the Turing machine accepts; otherwise, it rejects (for this guess of u, q and m). □

References

[1] J. Avenhaus and K. Madlener, "The Nielsen reduction and P-complete problems in free groups", *Theoretical Computer Science* **32** (1984) 61-76.

[2] J. Avenhaus and K. Madlener, "On the complexity of intersection and conjugacy problems in free groups", *Theoretical Computer Science* **32** (1984) 279-295.

[3] L. Babai, E. Luks, A. Seress, "Permutation groups in NC", *Proc. 19th ACM Annual Symposium on Theory of Computing* (1987) 409-420.

[4] J. Berstel, *Transductions and Context-Free Languages*, Teubner Verlag, Stuttgart (1979).

[5] C. Bennett, "Time/Space tradeoffs for reversible computation", *SIAM J. of Computing* **18** (1989) 766-776.

[6] J. Berstel, *Transductions and context-free languages*, Teubner, Stuttgart (1979).

[7] S. Cho, D. Huynh, "Finite-automaton aperiodicity is PSPACE-complete", *Theoretical Computer Science* **88** (1991) 99-116.

[8] D.E. Cohen, *Combinatorial Group Theory: A Topological Approach*, London Math. Soc. Student Texts **14**, Cambridge University Press (1989).

[9] D. Cowan, "Inverse monoids of dot-depth two", *Intern. J. Algebra and Computation* **3** (1993) 411-424.

[10] S. Eilenberg, *Automata, Languages and Machines*, Vol. B, Academic Press, New York (1976).

[11] M. Garey, D. Johnson, *Computers and Intractability: A Guide to the Theory of* NP-*Completeness*, Freeman, New York (1979).

[12] J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Formal Languages and Computation*, Addison-Wesley (1979).

[13] D. Kozen, "Lower bounds for natural proof systems", *Proc. 18th IEEE Annual Symposium on Foundations of Computer Science* (1977) 254-266.

[14] R. Levine, A. Sherman, "A note on Bennett's time-space tradeoff for reversible computations", *SIAM J. on Computing* **19** (1990) 673-677.

[15] R. Lyndon and P. Schupp, *Combinatorial group theory*, Springer (1977).

[16] S. Margolis, J. Meakin, "Free inverse monoids and graph immersions", *International J. of Algebra and Computation* **3** (1993) 79-100.

[17] R. McNaughton, S. Papert, *Counter-free Automata*, MIT Press (1971).

[18] M. Petrich, *Inverse monoids*, Wiley, New York, 1984.

[19] J.-E. Pin, *Variétés de langages formels*, Masson, Paris (1984). English translation: *Varieties of formal languages*, North Oxford, London (1986).

[20] K. Reidemeister, "Fundamentalgruppen und Überlagerungsräume", *Nachrichten der Ges. Wiss., Math. Phys. Klasse Göttingen* (1928) 69-76.

[21] R. Ruyle, Ph. Dissertation, University of Nebraska-Lincoln, in preparation.

[22] J. Sakarovitch, "A problem on rational subsets of the free group", *American Mathematical Monthly* **91** (1984) 499–501

[23] C. Sims, *Computation with Finitely Presented Groups*, Cambridge, London (1995).

[24] J. Stallings, "The topology of graphs", *Inventiones Mathematicæ* **71** (1983) 551–565.

[25] I.A. Stewart, "Refining known results on the generalized word problem for free groups", *International J. Algebra and Computation* **2** (1992) 221-236.

[26] J. Stern, "Complexity of some problems from the theory of automata", *Information and Control* **66** (1985) 163-176.

[27] P. Weil, *Inverse monoids and the dot-depth hierarchy*, Ph.D. Dissertation, University of Nebraska-Lincoln (1988).