ONLINE PRESENTATIONS OF FINITELY GENERATED STRUCTURES

NIKOLAY BAZHENOV, ISKANDER KALIMULLIN, ALEXANDER MELNIKOV, AND KENG MENG NG

Abstract. We systematically investigate into the online content of finitely generated structures. The online content of a potentially infinite algebraic or combinatorial structure is perhaps best reflected by its FPR-degrees (to be defined). We confirm a natural conjecture by showing that the FPR-degrees of a finitely generated structure must be dense. Remarkably, we show that FPR-degrees of a f.g structure does not have to be upwards dense. Finally, we disprove a natural conjecture about honestly generated structures (to be stated).

Keywords: finitely generated structure; isomorphism; primitive recursive function.

1. Introduction and results

What is the crucial difference between an online algorithm and an offline algorithm? In an online algorithm input is given piece-by-piece, and the algorithm can refer only to the part it has seen so far. In contrast, if our algorithm is offline then the input is given all at once. The classical complexity theory is focused around the study of the offline situation [GJ79, DF13]. For example, when we talk about the Chromatic Number problem [GJ79] in complexity theory, we assume that the whole of a graph is already given to us at once, and we must calculate the smallest number of distinct colours sufficient to cover the vertices of the graph so that no two adjacent vertices share the same colour. The problem is already NP-hard, but its online version is even more challenging. For example, it is clear that for a binary tree two colours suffice. However, the number of colours sufficient for online colouring can be shown to strictly increase with the size of a given tree [Kie98a]. In the case of finite structures most work comes from comparing offline vs online performance. The goal of online algorithm design is to improve what is called the Competitive Performance Ratio of online divided by offline. For example, first fit gives a competitive ratio of 2 for the classical Bin Packing problem [GJ79]; see [Kie98b].

Most of the results mentioned above are concerned with finite objects; that is, at the end the graph or the database is still assumed to be finite, and the algorithm is assumed to eventually stop. However, many modern online algorithms are dealing with massive, constantly changing and rapidly expanding databases, such as the World Wide Web. It is thus often natural to consider online computations with potentially infinite input. Beginning in the 1980's there has been quite a lot of work

2010 *Mathematics Subject Classification.* Primary 03D45, 03C57. Secondary 03D75, 03D80.
The authors were partially supported by Marsden Fund of New Zealand.

on online infinite combinatorics [Kie81, Kie98a, KPT94, LST89, Rem86]. Nonetheless, there is no systematic theoretical framework that would describe and explain the online situation for potentially infinite data. The present article contributes to the recently suggested framework [KMN17b, Mel17, BDKM] which aims to develop such a general theory. In this paper we concentrate on the natural case when the input is a finitely generated structure. For example, any term algebra over a finite alphabet is an example of a finitely generated structure; we note that term algebras play a significant role in the semantics of abstract data types.

From the purely algebraic point of view, finitely generated structures are often viewed as the structures which are understood best after the finite ones. For instance, there is a large body of research focused on algorithmic and purely algebraic aspects of finitely generated groups [Hig61, Gol64, NA68, Gro81, Ers12] and rings [AT51, Lew67, Nos83, AKNS]. One pleasant feature of such structures is that decision procedures in them are intrinsic, in the following sense. For example, if G is a finitely generated group and it has an algorithmically decidable word or conjugacy problem, then every $H \cong G$ will also have the problem decidable. Indeed, fix some generators $\bar{g}$ of G and their isomorphic counterparts $\bar{h}$ in H. Then every element of G is a word in the alphabet of $\bar{g}$, and it can be naturally mapped to the respective word in the alphabet of $\bar{h}$. This observation is heavily exploited in combinatorial group theory [Hig61, LS01], often without explicit reference.

Notice, however, that the observation above uses a rather general notion of an algorithmically effective process, namely a Turing computable process. For suppose we are given some algorithmic description of a group $\mathcal{C}$ which we know is isomorphic to $(\mathbb{Z}, +)$, and furthermore we know that c generates $\mathcal{C}$. Our job is to associate every $x \in \mathcal{C}$ with an integer. The naive algorithm would ask for a late enough stage at which we see $mc = x$, and then we can set $x \to m$. However, how long will we have to wait? It is easy to diagonalise against all polynomial time, all exponential, or even all hyper-exponential (etc.) algorithms. In other words, this procedure uses an instance of a *truly unbounded search*; this is the same as to say that the algorithm is not primitive recursive [Rog87].

It is natural to ask what happens if we forbid unbounded search. Of course, one naturally seeks to understand the truly efficient algorithms, in the spirit of, e.g., [KN95, BG00, KM10, CR91, CR98]. Kalimullin, Melnikov and Ng [KMN17b] have observed that making an algorithm merely primitive recursive is often sufficient to obtain a polynomial-time one, or even a finite automatic one [BHTK$^+$]. Removing unbounded search seems to be a crucial step whenever we try to transform a Turing computable procedure into a polynomial-time one. When we look at general primitive recursive algorithms – rather than, e.g., polynomial time, automatic, or linear ones – we strip away much of the irrelevant counting combinatorics. Thus, primitive recursion serves as a *useful abstraction*. All that matters is that there is *some* precomputed bound on our searches. The effects of this seemingly relaxed restriction have proven to be rather significant. This paper contributes to the general program that systematically investigates into these effects which have never been seen before in computable algebra or combinatorics; see [KMN17b, Mel17, BDKM, DHTK$^+$, KMN17a, MN].

To state our results, we need formal definitions.

1.1. **Punctual presentations and punctual degrees.** Kalimullin, Melnikov, and Ng [KMN17b] proposed that an "online" algebraic structure must minimally satisfy the following general definition.

Definition 1.1. A countable structure is *fully primitive recursive* (*fpr*) if its domain is $\mathbb{N}$ and the operations and predicates of the structure are (uniformly) primitive recursive.

The main intuition is that we need to define more of the structure "without delay". Here "delay" really means an instance of a truly unbounded search. We informally call fpr structures *punctually computable* or simply *punctual*. We could also agree that all finite structures are also punctual by allowing initial segments of $\mathbb{N}$ to serve as their domains. Although the definition above is not restricted to finite languages, we will never consider infinite languages in the article.

Recall that the inverse of a primitive recursive function does not have to be primitive recursive. Fix a punctual structure $\mathcal{A}$. The collection of all punctual presentations of $\mathcal{A}$ carries a natural *reduction*, as defined below.

Definition 1.2. Let $\mathcal{A}$ be a punctual structure. Then, for punctual $\mathcal{C}, \mathcal{B}$ isomorphic to $\mathcal{A}$,

$$\mathcal{C} \leq_{pr} \mathcal{B} \text{ if there exists a surjective primitive isomorphism } f : \mathcal{C} \to_{onto} \mathcal{B}.$$

This leads to an equivalence relation $\cong_{fpr}$ and the degree structure on the classes which will be denoted $\mathbf{FPR}(\mathcal{A})$.

What does $\mathbf{FPR}(\mathcal{A})$ reflect? If $\mathcal{C} \leq_{pr} \mathcal{B}$ then, in a way, $\mathcal{B}$ has more online content than $\mathcal{C}$ does, in the sense that more things happen in $\mathcal{B}$. In other words, $\mathcal{C} \leq_{pr} \mathcal{B}$ means that $\mathcal{B}$ enumerates itself more impatiently.

For example, the standard copy of $(\mathbb{Q}, <)$ punctually embeds any other punctual copy of the rationals; it has a prompt Skolem function, but some other copies may have slow intervals. The dense linear order is a canonical example when a computable back-and-forth method works, the other common (algebraically) homogeneous examples include the random graph and the Fraïsse limit of finite abelian p-groups. Remarkably, the FPR-degrees of the dense linear order, the random graph, and the universal abelian p-group are pairwise non-isomorphic; see [MN].

We go back to the finitely generated case. In contrast with $(\mathbb{Q}, <)$, the standard copy of (ω, S) can be punctually embedded into any other copy. Any other copy of (ω, S) will contain points that look non-standard ("infinite", "disconnected") for a very long time. It is not difficult to show that $\mathbf{FPR}(\omega, S)$ is dense, infinite, and furthermore upwards dense.

The reader should take some time and try to come up with an example of a finitely generated (infinite) structure whose FPR-degrees would not have these three properties: dense, infinite, and upwards dense. Indeed, all natural examples – including the elementary f.g. groups, monoids, and unary structures – seem to satisfy these properties. What could prevent us from making an element "look non-standard" or "far-far away" from the finitely many generators? Nonetheless, Kalimullin, Melnikov, and Ng [KMN17b] have constructed an example of a finitely generated punctual structure $\mathcal{A}$ with the property $|\mathbf{FPR}(\mathcal{A})| = 1$. (We will show that, for a f.g. $\mathcal{A}$, $|\mathbf{FPR}(\mathcal{A})| = 1$ is equivalent to saying that every two punctual

copies of the f.g. $\mathcal{A}$ are isomorphic via a primitive recursive f having primitive recursive inverse; see Fact 2.1.)

So $\mathbf{FPR}(\mathcal{A})$ does not have to be infinite, let alone upwards dense, for an infinite f.g. $\mathcal{A}$. Can we have $1 < |\mathbf{FPR}(\mathcal{A})| < \infty$? The first result of the paper implies that the answer is negative, because it confirms the density conjecture:

Theorem 1.3. *Suppose a punctual $\mathcal{A}$ is finitely generated. Then $\mathbf{FPR}(\mathcal{A})$ is dense.*

Recall that we started with testing three properties: infinite, dense, upwards dense. Although we know that $|\mathbf{FPR}(\mathcal{A})| = 1$ is possible, we have confirmed the density conjecture. Suppose $\mathbf{FPR}(\mathcal{A})$ is infinite, then is $\mathbf{FPR}(\mathcal{A})$ upwards dense? Rather surprisingly, this natural hypothesis fails in the strong sense below.

Theorem 1.4. *There exists a f.g. $\mathcal{A}$ such that $|\mathbf{FPR}(\mathcal{A})| = \infty$ and $\mathbf{FPR}(\mathcal{A})$ has a greatest element.*

The proof of the theorem above is combinatorially quite involved. It generalises a strategy introduced in [KMN17b] and blends it with a new technique. The high combinatorial complexity of the argument leaves some hope. More specifically, perhaps the upward density hypothesis will hold for many general enough natural classes of finitely generated structures.

Problem 1.5. *Give a general enough sufficient condition on a finitely generated $\mathcal{A}$ which would imply that $|\mathbf{FPR}(\mathcal{A})|$ is upward dense.*

Although Theorem 1.4 disproves a natural conjecture, the authors suspected that the upward density might fail via a non-trivial construction. However, initially the authors were almost certain that another natural enough conjecture below must be provable.

To state (and disprove!) the conjecture, we need another definition. But first, we give some intuition. Note that adding the predecessor unary function P into the language of (ω, S) will not result in $|\mathbf{FPR}(\omega, S, P)| = 1$. We could still keep an element looking "non-standard" for as long as necessary. Adding Scolem function for division or "$-$" (or both) into the language of finitely generated abelian groups does not seem to have any significant effect on their FPR-degrees either. More generally, even if we could quickly find the elements that generate a given element via a given term (if they exist), it seemingly would not give us any extra power. It seems that the temporarily non-standard looking "islands" will still be "far-far away" even if we allow to promptly generate the structure backwards.

Definition 1.6. Let $\mathcal{A}$ be a punctual structure in a finite functional language. We say that $\mathcal{A}$ is *honestly generated* if there is a primitive recursive procedure which, for every term t and each element $x \in \mathcal{A}$:

(1) decides whether $\exists \bar{y}\, t(\bar{y}) = x$, and
(2) if the answer is "yes", it gives such a $\bar{y}$.

(The definition above is a bit weaker than punctual 1-decidability which was defined in [KMN17b].)

Our initial conjecture was: Suppose $|\mathbf{FPR}(\mathcal{A})| = \infty$ for a finitely generated $\mathcal{A}$, then $\mathcal{A}$ has infinitely many honestly generated copies, up to $\cong_{pr}$. Following the general pattern of this paper, we refute this conjecture:

Theorem 1.7. *There exists a f.g. structure $\mathcal{A}$ such that $|\mathbf{FPR}(\mathcal{A})| > 1$ (thus, $|\mathbf{FPR}(\mathcal{A})| = \infty$) and $\mathcal{A}$ has a unique honestly generated punctual presentation, up to $\cong_{pr}$.*

We note that uniqueness up to $\cong_{pr}$ is the same as uniqueness up to primitive recursive isomorphism with primitive recursive inverse, by Fact 2.1. The proof is again non-trivial, but the good news is that it shares one strategy with the proof of Theorem 1.4. This common feature will allow for a more compact exposition. Also, as before, there is perhaps some general enough sufficient condition that implies the conjecture.

We leave open if being honestly generated can be replaced by punctual 1-decidability in the theorem above. See [BDKM, Mel17, KMN17b] for a detailed discussion of punctual 1-decidability. Finally, we leave open whether the counterexamples constructed in this paper can be witnessed by finitely generated groups or at least semigroups.

2. The first steps. Proof of Theorem 1.4

All algebraic structures throughout the paper are finitely generated and have a finite functional language.

It is clear that every finitely generated punctual structure has the least presentation under $\leq_{pr}$. This least presentation is the naturally generated term algebra upon some (any) fixed tuple of generators. We usually write $\mathcal{B}$ for this smallest presentation, where $\mathcal{B}$ stands for "bottom". We note that the choice of the generators does not really matter here, for we could primitively recursively bi-embed two different naturally generated term algebras onto one another.

It is easy to construct an example of a (not finitely generated) pair of punctual structures $\mathcal{A}$ and $\mathcal{B}$ such that $\mathcal{A} \leq_{pr} \mathcal{B}$ and $\mathcal{B} \leq_{pr} \mathcal{A}$ but $\mathcal{B} \not\cong_{fpr} \mathcal{A}$ in the sense that there is no primitive recursive isomorphism with primitive recursive inverse between them; we omit details. It is not known, however, if $|\mathbf{FPR}(\mathcal{A})| = 1$ is equivalent to $\mathcal{A}$ being punctually categorical. In [MN], Melnikov and Ng used a rather non-trivial argument to illustrate that the two notions are equivalent for undirected graphs. In contrast, the case of finitely generated structures is rather straightforward.

Fact 2.1 (With Harrison-Trainor)**.** *Suppose $\mathcal{A}$ is a finitely generated punctual structure, and suppose $|\mathbf{FPR}(\mathcal{A})| = 1$. Then $\mathcal{A}$ is punctually categorical.*

Proof. Let $\mathcal{B}$ be the natural presentation of $\mathcal{A}$ which is the term algebra over generators $\bar{b}$. Let $\mathcal{C}$ be some other punctual copy of $\mathcal{A}$, and let $f : \mathcal{B} \to \mathcal{C}$ be a primitive recursive isomorphism from $\mathcal{B}$ onto $\mathcal{C}$. Write $\bar{b}'$ for the f-image of $\bar{b}$ in $\mathcal{C}$. Fix a primitive recursive surjective isomorphism $g : \mathcal{C} \to \mathcal{B}$, and let $\bar{b}''$ be the g-image of $\bar{b}'$ in $\mathcal{B}$. Since $\bar{b}''$ generates $\mathcal{B}$, it must generate $\bar{b}$; fix the finitely many terms witnessing this fact.

Consider $c \in \mathcal{C}$. We explain how to promptly compute $f^{-1}(c)$. Compute $g(c) \in \mathcal{B}$. Every element of $\mathcal{B}$ is a term in $\bar{b}$ and, thus, is a term in $\bar{b}''$; the latter term can be quickly computed using the finitely many terms that witness that $\bar{b}$ is generated by $\bar{b}''$. Recall that $\bar{b}'' = g(\bar{b}')$. If $g(c) = t(\bar{b}'')$ then $c = t(\bar{b}')$. Recall that $\bar{b}' = f(\bar{b})$ and set $f^{-1}(c) = t(\bar{b})$. □

2.1. **Proof of Theorem 1.4.** We need to prove that $\mathbf{FPR}(\mathcal{A})$ is dense. If $|\mathbf{FPR}(\mathcal{A})| = 1$ then there is nothing to prove. Otherwise, let $\mathcal{B} <_{pr} \mathcal{T}$ be two punctual presentations of $\mathcal{A}$, where $\mathcal{B}$ is not necessarily the "bottom" copy of $\mathcal{A}$. We build a punctual

copy $\mathcal{X}$ with the property:

$$\mathcal{B} <_{pr} \mathcal{X} <_{pr} \mathcal{T}.$$

We meet the requirements:

$$P_e : p_e : \mathcal{X} \to \mathcal{B} \text{ is not an onto isomorphism;}$$
$$R_j : p_j : \mathcal{T} \to \mathcal{X} \text{ is not an onto isomorphism,}$$

and we also construct two surjective primitive recursive isomorphisms $f : \mathcal{B} \to \mathcal{X}$ and $g : \mathcal{X} \to \mathcal{T}$.

The idea is to switch between copying $\mathcal{B}$ and $\mathcal{T}$. However, this would not be possible without $\mathcal{A}$ being finitely generated. Fix the generators $\bar{g}$ of $\mathcal{B}$. Recall that $\mathcal{B} <_{pr} \mathcal{T}$, and assume h witnesses this reduction. We let $\bar{g}''$ be the h-image of $\bar{g}$ in $\mathcal{T}$. We will also initially define f to be the identity map naturally copying $\mathcal{B}$ into $\mathcal{X}$ for a few stages of the construction. Let $\bar{g}'$ be the image of $\bar{g}$ in $\mathcal{X}$.

The strategy for P_e. In the construction, we will initially start by (globally) waiting until we see that $p_e(\bar{g}')$ generates $\bar{g}$ in $\mathcal{B}$. If this never happens then p_e cannot be an onto isomorphism, and thus the requirement will be met. Suppose we see that $p_e(\bar{g}')$ generates $\bar{g}$ in $\mathcal{B}$. Then the strategy is declared *ready for diagonalisation*.

Assume that P_e is the highest priority requirement and has been declared ready for diagonalisation. If $\mathcal{X}$ is currently naturally copying $\mathcal{B}$ then *switch* to copying $\mathcal{T}$ (to be explained below). Then wait for a finitary disagreement in p_e illustrating that p_e is not an isomorphism. (In the construction, the action of switching will be decided according to the priority order.)

The strategy for R_j. The same as for P_e above, mutatis mutandis.

The switching. There are two substantially different cases that need to be treated separately.

Switching from $\mathcal{B}$ to $\mathcal{T}$. Suppose $\mathcal{X}$ is currently naturally copying $\mathcal{B}$ via f, and assume we are willing to switch to copying $\mathcal{T}$ into $\mathcal{X}$. To perform the switch, identify $\mathcal{B}$ (and, thus, $\mathcal{X}$) with $h(\mathcal{B})$ in $\mathcal{T}$. From now onwards, change the interpretation of the currently computed atomic diagram of $\mathcal{X}$ by replacing each natural pre-image of $x \in \mathcal{X}$ in $\mathcal{B}$ by the respective element in $\mathcal{T}$.

Switching from $\mathcal{T}$ to $\mathcal{B}$. Now suppose $\mathcal{X}$ is currently copying $\mathcal{T}$, and we need to switch to $\mathcal{B}$ at stage s. Stop copying $\mathcal{T}$ but keep producing the term algebra generated by $\mathcal{X}_s$ which is naturally associated with the part $\mathcal{T}_s$ of $\mathcal{T}$ that had been copied into $\mathcal{X}$ before stage s. Naturally identify these elements with the terms of the respective g-images in $\mathcal{T}$. Wait for $h(\mathcal{B})$ to catch up with $\mathcal{T}_s$ by generating all elements in $\mathcal{T}_s$. (Note that this must eventually happen because the structure is finitely generated and h is onto.)

As soon as we see that $h(\mathcal{B})$ has covered all of $\mathcal{T}_s$, pause the enumeration of $\mathcal{X}$ and quickly compute all the terms that we have used in the enumeration of $\mathcal{X}$ since stage s for $h^{-1}(\mathcal{T}_s)$ in $\mathcal{B}$. Note that there is a natural bound on the time needed to make these computations. This will give us a natural pre-image for the currently constructed part of $\mathcal{X}$. Switch to $\mathcal{X}$ copying $\mathcal{B}$ by identifying (the currently built part of the diagram of) $\mathcal{X}$ with this natural pre-image in $\mathcal{B}$.

Construction. We pick some natural priority ordering on the requirements. If there are no requirements that are ready for diagonalisation then copy $\mathcal{B}$ into $\mathcal{X}$. Otherwise, fix the highest priority requirement which is ready for diagonalisation and switch $\mathcal{X}$ (if necessary) until the requirement is met.

Verification. Much of the verification was incorporated into the description of the diagonalisation strategies and the switching procedure. We only clarify a few points which were not explained in detail above. The description of the switching strategy guarantees that $\mathcal{X}$ is primitive recursive and is isomorphic to $\mathcal{A}$. Each diagonalisation requirement P_e (and R_j) is met because otherwise we'd have a contradiction with $\mathcal{B} <_{pr} \mathcal{T}$. It is also crucial that the disagreement will be discovered at a finite stage and will be finitary in nature. This is because we used the generators to ensure that if p_e (or p_j) is an isomorphism then it must be onto; see the description of P_e.

3. Proof of Theorem 1.4

Proof idea. The proof blends a strategy from [KMN17b] with a new diagonalisation strategy; we briefly describe the main idea of both strategies below. As was proven in [KMN17b], one can build a rigid 1-generated punctually categorical structure, as follows. Start building $\mathcal{A}$ as an ω-chain using a unary function s, and also start adjoining loops of specific sizes to the chain using another unary function c. Also, fix a third unary function p that maps points back to the fixed generator. To make sure every punctual $\mathcal{P} \cong \mathcal{A}$ is punctually isomorphically mapped onto $\mathcal{A}$, choose the sizes of the loops thoughtfully. We could, for example, start by producing only 1-loops and wait for $\mathcal{P}$ to respond by giving two consequent 1-loops. As soon as $\mathcal{P}$ responds, we stop producing only 1-loops and start putting pairs consisting of a 1-loop and a 2-loop. Then, relative to $\mathcal{P}$, we can use p (and the time it took $\mathcal{P}$ to respond) to reconstruct the exact location of its 1-loop and match it with the respective one in $\mathcal{A}$. If $\mathcal{P}$ gives us a loop of a wrong size we can forbid this size in the construction. Also, if $\mathcal{P}$ gives us a chain which is too long, we will keep the sizes of our loops smaller than this chain, thus potentially forcing it to grow to infinity, so $\mathcal{P} \ncong \mathcal{A}$. There are several further subtleties in the construction of $\mathcal{A}$, see [KMN17b] and [BDKM] for further intuition. For instance, one must be careful in the choice of loop patterns to make sure that the ith punctual structure can be pressed essentially independently from the other ones. See the formal description of the pressing strategy below for further details.

Remarkably, the above strategy is consistent with the (ω, S)-style diagonalisation. In (ω, S), we can keep an element disconnected from the origin 0 for as long as we want. We can use this non-standard-looking element to diagonalise against an isomorphism from our copy to the canonical copy of (ω, S). In our $\mathcal{A}$ we will be keeping one extra chain of loops disconnected from the origin. A careful choice of sizes of loops and of a generalised predecessor function will still allow us to press $\mathcal{P}$ despite of the fact that the extra chain is currently not connected to the generator. Indeed, it is sufficient for us to locate the exact coordinate of a point in the chain in $\mathcal{P}$ and match it correctly and promptrly with the respective point in $\mathcal{A}$. To do so we do not have to really go all the way back to the generator. We are ready to give the formal details.

Proof. The language of the structure $\mathcal{A}$ contains the following symbols:

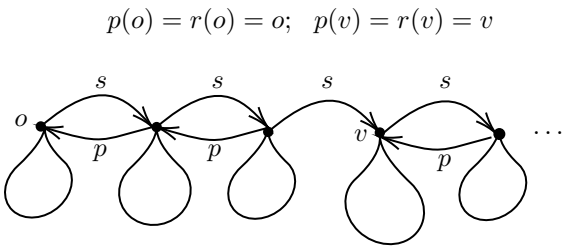

FIGURE 1. The structure $\mathcal{A}$ with two roots o and v shown.

- a constant o, and
- unary functions s, c, p, and r.

The structure $\mathcal{A}$ (see Figure 1) will have the following properties:

(1) The domain of $\mathcal{A}$ is a disjoint union of finite c-cycles. For any c-cycle C, the function s maps every element from C to a fixed element from another c-cycle C'. The function s is arranged in such a way that the values o, $s(o)$, $s(s(o))$, ..., $s^n(o)$, $s^{n+1}(o)$, ... form an ω-chain, and every c-cycle from $\mathcal{A}$ contains exactly one element of the form $s^k(o)$, $k \in \omega$.

(2) For $x \in \omega$, consider the unique number n such that $s^n(o)$ belongs to the c-cycle of x. Then $p(x)$ satisfies one of the following two cases:
   (a) $p(x) = s^{n-1}(o)$, or
   (b) $p(x) = s^n(o)$.
   Furthermore, $p(c^k(x)) = p(x)$, for any $k \in \omega$.
   Consider the greatest $m \leq n$ such that $p(s^m(o)) = s^m(o)$. Such a number m exists, since $p(o) = o$. Then we have $r(x) = s^m(o)$.

(3) The element $y \in \mathcal{A}$ is called a *root* if $r(y) = y$ or, equivalently, if $p(y) = y$. Notice that for every root y, there is a natural number n such that $y = s^n(o)$. For an element x, the *island coordinates* (m, k) of x are defined as follows: m is the number such that $s^m(r(x))$ belongs to the c-cycle of x; and k is the least number such that $x = c^k(s^m(r(x)))$.

Note that the first property above ensures that $\mathcal{A}$ is rigid and 1-generated. The *canonical presentation* of $\mathcal{A}$ is built as the term algebra generated by o. The canonical fpr presentation of $\mathcal{A}$ will be denoted by $\mathcal{B}$ ("bottom"). We will also build an fpr copy $\mathcal{T}$ ("top") of $\mathcal{A}$.

We use the pairing function $\langle n, m \rangle = 2^n \cdot (2m + 1) - 1$.

We fix a uniformly computable list of all primitive recursive unary functions $\{h_e\}_{e \in \omega}$. We also choose a uniformly computable list of all fpr structures in the language of our structure $\mathcal{A}$:

$$\mathcal{C}_n = (\omega; o_n, s_n, c_n, p_n, r_n), \ n \in \omega.$$

The functions s_n, c_n, p_n, r_n are treated as partial computable functions with corresponding primitive recursive time functions t_n: we assume that the value $f(x)[t_n(x)]$ converges, for any $n, x \in \omega$ and $f \in \{s_n, c_n, p_n, r_n\}$.

We will satisfy the following series of requirements:

P_e: h_e is not an isomorphism from $\mathcal{T}$ onto $\mathcal{B}$.
R_n: If $\mathcal{C}_n \cong \mathcal{T}$, then there is a primitive recursive isomorphism from $\mathcal{C}_n$ onto $\mathcal{T}$.

For a functional symbol $f \in \{s, c, p, r\}$, we use notations f_B and f_T to denote the interpretation of the symbol f inside $\mathcal{B}$ and $\mathcal{T}$, respectively.

An overview.

The structures $\mathcal{B}$ and $\mathcal{T}$ are constructed with the following agreement in mind. One may think of $\mathcal{T}$ as a version of $\mathcal{B}$ which runs a little bit faster than $\mathcal{B}$: At the beginning of each stage s, $\mathcal{T}[s]$ consists of two disjoint parts B' and I, where B' is isomorphic to $\mathcal{B}[s]$, and I is a (possibly empty) substructure of $\mathcal{T}[s]$ such that (for now) I does not contain elements of the form $s_T^k(o)$, $k \in \omega$. Informally speaking, B' is copying $\mathcal{B}$, and I is an *island*. The island I will be eventually attached to B' with the help of the function s_T (to be explained below).

The priority ordering of the strategies is arranged as per usual: $R_0 < P_0 < R_1 < P_1 < \dots$. Every strategy P_e will be in one of the following states:

(S1) unstarted;
(S2) calculating the island tag;
(S3) active;
(S4) finished.

An unstarted strategy P_e wants to begin building the P_e-*island* inside $\mathcal{T}$. First, P_e goes into state S2 and (slowly) calculates the length l_e of the c-cycle for the root of the P_e-island. We call this l_e the *island tag* of P_e. Note that these calculations involve evaluating various things inside structures $\mathcal{C}_i$, $i \leq e$. When the island tag is computed, P_e goes into state $S3$.

When P_e is active, it builds its own island while looking for an opportunity to diagonalize against h_e. When this opportunity is seized, P_e becomes finished.

Note that at a stage s, each of the strategies R_i, $i \leq s$, can construct something in $\mathcal{B}$ and $\mathcal{T}$.

In order to ensure that both $\mathcal{B}$ and $\mathcal{T}$ are punctual, at each stage s of the construction we proceed as follows:

- pick the least k such that $s_B^k(o)$ is still undefined, and define it as the next element in the domain;
- introduce the c_B-cycle of $s_B^k(o)$ according to the description of the strategies (see below);
- the functions p_B and r_B on the new elements are defined right away;
- the structure $\mathcal{T}$ copies the new elements from $\mathcal{B}$.

Beforehand, we assign to each P_e, $e \in \omega$, the set of *acceptable tags*

$$L(P_e) = \{m_0(e), m_1(e), \dots, m_e(e), m_{e+1}(e)\}.$$

We ensure that for $i \neq j$, $L(P_i)$ and $L(P_j)$ are disjoint: e.g., one can define

$$m_k(e) := 3\langle e+1, k\rangle + 3.$$

A strategy P_e will always choose its island tag as one of the elements from $L(P_e)$.

We assume that o belongs to a c-cycle of size 3, and we set $L(P_{-1}) := \{3\}$. Thus, one can say that the main part of $\mathcal{T}$ is tagged by the length 3.

The description of P_e in isolation. We note that the strategy will have to be slightly modified in the construction.

Recall that P_e has the finite set of acceptable tags $L(P_e)$. Each higher priority strategy R_n, $n \leq e$, can *forbid* to use at most one tag l from $L(P_e)$. Therefore, the choice of the cardinality of $L(P_e)$ ensures that at any stage s, the strategy P_e has at least one *non-forbidden* tag.

First, we pick the next element w_e in the domain of $\mathcal{T}$ and the least non-forbidden tag $l_e \in L(P_e)$. The length l_e is declared the *island tag* of P_e.

While the value $h_e(w_e)[t]$ either is undefined or does not belong to $\mathcal{B}[t]$, build the P_e-*island* inside $\mathcal{T}$ as follows (note that we always use next elements from the domain to build islands):

- Put w_e in a c-cycle of size l_e. The element w_e will be the root of the P_e-island: for any element x from the island, we will have $r(x) = w_e$.
- Continue constructing the island as follows. Suppose that $s^k(w_e)$ is already defined. Then we put a fresh c-cycle C (its length is decided by the R_i-strategies) and connect it in a natural way with our island, i.e. we declare some element from C as $s^{k+1}(w_e)$, and for each $x \in C$, define $p(x) = s^k(w_e)$ and $r(x) = w_e$.

Suppose that t_0 is the first stage such that $h_e(w_e)[t_0]{\downarrow} \in \mathcal{B}[t_0]$. Then we attach the island to the main part of $\mathcal{T}[t_0]$ and extend $\mathcal{B}[t_0]$ to an isomorphic copy of $\mathcal{T}$. More formally, let k be the least number such that the value $s^k_B(o)$ is still undefined at stage t_0. We define:

(i) $s^k_T(o) := w_e$;
(ii) $s^k_B(o)$ is the least unused element in $\mathcal{B}$, and we grow the main part of $\mathcal{B}$ by attaching a copy of the constructed P_e-island.

Then requirement P_e is declared *satisfied*: Indeed, our construction will be organized in such a way that before stage t_0, the structure $\mathcal{B}[t]$ does not contain c-cycles of size l_e. Therefore, the element $h_e(w_e)$ cannot belong to a c_B-cycle of size l_e, and h_e cannot be an isomorphism from $\mathcal{T}$ onto $\mathcal{B}$.

After that, every new c-cycle inside $\mathcal{B}$ will be attached to the "end" of the "glued" copy of the P_e-island.

The description of R_e in isolation.

First, consider the element $x = 0$ from $\mathcal{C}_e$. We simultaneously proceed with the following procedures:

(A) Find the least number l such that $c^l_e(r_e(x)) = r_e(x)$ (i.e. the length of the c_e-cycle of the root $r_e(x)$).
(B) Apply p_e to x at most 2^{e+1} times to find some y_0 associated with a c_e-cycle of size $3e + 1$. When such y_0 is found, compute $y_1 = p_e^{2^{e+1}}(y_0)$. Check whether y_1 lies in a c_e-cycle of size $3e + 1$.

While we are acting as decribed above, we cannot delay the construction of $\mathcal{B}$ and $\mathcal{T}$. Hence, we build $\mathcal{B}$ and $\mathcal{T}$ by adding new elements. If a newly added x has island coordinates $(\langle e, k\rangle, 0)$ for some k, then we build its c-cycle as a new cycle of size $3e+1$. Note that a new element x from $\mathcal{T}$ can appear because of either copying $\mathcal{B}$, or building a P_i-island for some i.

Let u_0 be the number of steps needed to finish the procedures above. The described actions can have one of the following outcomes:

(i) One of the procedures (A) or (B) never stops. Then $\mathcal{C}_e$ is not isomorphic to $\mathcal{T}$.

(ii) Otherwise, both procedures eventually stop. Suppose that the number l does not belong to any $L(P_i)$, where $-1 \leq i < e$. Then we will ensure that $\mathcal{C}_e \not\cong \mathcal{T}$: If $l \in L(P_k)$ for some $k \geq e$, then the strategy R_e will *forbid* the tag l.

(iii) Assume that the number l lies in some $L(P_i)$, $-1 \leq i < e$.

(iii.a) If $\mathcal{C}_e \cong \mathcal{T}$ and we prematurely reached the root $r_e(x)$ by applying p_e, then we can quickly find the island coordinates of $x = 0$.

(iii.b) Otherwise, we found two adjacent c_e-cycles of size $3e + 1$. Then we *switch* to the next pattern (see below).

If u_0 is never calculated, then $\mathcal{C}_e$ is not isomorphic to $\mathcal{T}$. After u_0 is calculated, we will ensure that $\mathcal{T}$ has no pair of adjacent c_T-cycles of size $3e + 1$.

Now we consider the element $x = 1$ from $\mathcal{C}_e$. Again, we simultaneously proceed with the following:

(A′) Find the length l_1 of the c_e-cycle of the root $r_e(x)$.

(B′) Apply p_e to x at most 2^{e+1} times to find some y_0 associated with a c_e-cycle of size $3e + 1$. When such y_0 is found, compute $y_1 = p_e^{2^{e+1}}(y_0)$. After that, compute $y_2 = p_e^{2^{e+1}}(y_1)$. Check whether y_1 lies in a c_e-cycle of size $3e + 2$ and y_2 belongs to a cycle of size $3e + 1$. In other words, we *search for a pattern* $[3e + 1, 3e + 2, 3e + 1]$.

While we are acting as decribed above, we continue building $\mathcal{B}$ and $\mathcal{T}$. For new elements x with island coordinates $\langle e, k\rangle$, where $k \geq u_0$, we alternate between the c-cycles of size $3e + 1$ and $3e + 2$.

Let u_1 be the number of steps needed to finish the procedures (A′) and (B′). There will be one of the following outcomes:

(i′) One of the procedures (A′) or (B′) never stops. Then $\mathcal{C}_e$ is not isomorphic to $\mathcal{B}$.

(ii′) The number l_1 does not belong to any $L(P_i)$, where $-1 \leq i < e + 1$. If $l_1 \in L(P_k)$ for some $k \geq e + 1$, then R_e forbids the tag l_1.

(iii′) Otherwise, the tag l_1 lies in some $L(P_i)$, $-1 \leq i < e + 1$.

(iii′.a) If $\mathcal{C}_e \cong \mathcal{T}$ and we prematurely reached the root $r_e(x)$, then we can quickly determine the island coordinates of $x = 1$.

(iii′.b) Otherwise, we found a c_e-pattern $[3e+1, 3e+2, 3e+1]$. We will switch to the pattern $[3e + 1, 3e + 2, 3e + 2, 3e + 1]$.

After that, we make similar actions for the number $x = 2$: While searching for the length l_2 of the c_e-cycle of $p_e(2)$ and a pattern $\widehat{P} := [3e+1, 3e+2, 3e+2, 3e+1]$ inside $\mathcal{C}_e$, we iterate the pattern $\widehat{P}$ (in appropriate way) inside $\mathcal{B}$ and $\mathcal{T}$. The number u_2 is the number of steps needed to finish this search. When this is finished, we want to treat l_2 as a non-forbidden tag either for the main part of $\mathcal{T}$, or for one of the P_i-islands, where $i < e + 2$. If such a treatment is impossible, then we will ensure that $\mathcal{C}_e$ is not a copy of $\mathcal{T}$. The acquired tag l_2 helps us to find (the location of) the root for x. We switch to the next pattern $[3e+1, 3e+2, 3e+2, 3e+2, 3e+1]$.

The calculations of $u_0, u_1, u_2, \ldots$ allow us to build a primitive recursive isomorphism from $\mathcal{C}_e$ onto $\mathcal{T}$ (to be elaborated below).

Construction.

We view every considered primitive recursive function as a partial computable function $\varphi_e(x)$ such that the computation of $\varphi_e(x)$ converges in $p(x)$ many steps

for some primitive recursive p. The structure $\mathcal{C}_e[s]$ *evaluated at stage* s refers to a finite substructure of $\mathcal{C}_e$ on the domain $\{0, 1, \dots, s\}$ with all functions evaluated up to s steps. At stage s we are allowed to use only structures $\mathcal{C}_e[s]$, $e \leq s$.

Our construction defines partial computable functions s, c, p, r for $\mathcal{B}$ and for $\mathcal{T}$ and ensures that all the values $s(x), c(x), p(x), r(x)$ converge within x many steps of the construction. Therefore, both $\mathcal{B}$ and $\mathcal{T}$ are punctual.

Every strategy P_e can have a finite set of labels of the form $\boxed{forb(i, l)}$, where $i \leq e$ and $l \in L(P_e)$. A label $\boxed{forb(i, l)}$ means that the strategy R_i forbids our P_e to use the tag l. For every $i \leq e$, P_e can be labelled by at most one label $\boxed{forb(i, l)}$, $l \in \omega$. Note that we never remove added labels.

At stage s we assume that every strategy P_j, where $j > s$, is still *unstarted*.

Phase 0: Dealing with the strategies P_e.

(0.1) Declare the strategy P_s *calculating the island tag*, i.e. P_s goes from state S1 into S2.

(0.2) Consider the strategies $P_0, P_1, \dots, P_s$ in turn. Suppose that a strategy P_i, $i \leq s$, is in state S2. Then we look at numbers $j \leq i$, in turn. If P_i has no label of form $\boxed{forb(j, l_1)}$ (i.e. R_j has not forbidden a tag yet), then for each acceptable tag $l \in L(P_i)$, make $(s + 1)$ steps of computation of the following procedure:

$PROC_{i,j,l}$: For each element $x \leq s$ (in turn), calculate the values $r_j(x)$, $c_j(r_j(x))$, $c_j^2(r_j(x)), \dots, c_j^l(r_j(x))$. If for some x, the corresponding root $r_j(x)$ belongs to a c_j-cycle of size l, then immediately stop the procedure and output "forbid l." Otherwise, when all calculations are finished, output "allow l."

If the computation finishes (in $s + 1$ steps) with an output "forbid l," then put the label $\boxed{forb(j, l)}$ onto P_i.

By $PROC^0_{i,j,l}$ we denote the following modification of the procedure $PROC_{i,j,l}$: In place of all $x \leq s$, we consider only $x \leq i - j$.

(0.3) Let P_i be the least strategy which is currently in state S2. Suppose that M is the maximal element from $L(P_i)$.

Make $(s + 1)$ steps of computation of the following procedure:

> Compute a finite set F of *forbidden tags* as follows: For each $j \leq i$, search for the least $l \in L(P_i)$ such that the procedure $PROC^0_{i,j,l}$ outputs "forbid l." If there is such a tag l, then put this l into F. Otherwise, check whether P_i now has a label $\boxed{forb(j, l')}$: If there is such a label, then add the corresponding l' into F.

W.l.o.g., one may assume that the computation above does not stop until it calculates all values $c_j^M(r_j(x))$, for $j \leq i$ and $x \leq i - j$.

If the computation does not finish in $s + 1$ steps, then P_i stays in state S2.

Otherwise, find the least tag $l \in L(P_i) \setminus F$ (such l exists, since F contains at most $i + 1$ elements). We declare l the *island tag* of P_i, and say that P_i is now in state S3 (i.e. active). We initialize P_i by building the P_i-*island* $I[P_i; s]$ (inside $\mathcal{T}[s]$) as a c_T-cycle of size l. We choose an element w from the cycle and declare it the P_i-*witness*.

(0.4) If there is at least one active strategy P_i, then proceed to Phase 1. Otherwise, proceed straight to Phase 2.

Phase 1: Satisfying active strategies. Consider each active strategy P_e, in turn. Suppose that w is the P_e-witness.

If the value $h_e(w)$ is calculated in $(s+1)$ steps of computation and $h_e(w)$ belongs to $\mathcal{B}[s]$, then declare P_e *satisfied* and *finished*. Attach the P_e-island to the main part of $\mathcal{T}[s]$ as follows: Let k be the least number such that $s^k_B(o)$ is still undefined. Declare $s^k_T(o) := w$ and define the other values of s_T appropriately. After that, extend the structure $\mathcal{B}[s]$ in such a way that the resulting structure is isomorphic to the current $\mathcal{T}[s]$.

Otherwise, let m be the least number such that $s^m_T(w)$ is still undefined. Define $s^m_T(w)$ as the next element of the domain. Grow the c_T-cycle of the element according to the current working patterns (to be explained in Phase 2).

After that proceed to Phase 2.

Phase 2: Growing the structure $\mathcal{B}$. First, we explain how we define our working patterns.

Let $u_{q,l,e}(x)$ be the number of steps needed to perform the following effective procedure inside the structure $\mathcal{C}_e$:

(i) Find the value $r_e(x)$. If $r_e(x) \neq q$, then we stop *successfully*.
(ii) Otherwise $r_e(x) = q$. Compute the values $c^k_e(q)$, for $k \leq l$. If $c^l_e(q) \neq q$ or there is a non-zero number $k < l$ with $c^k_e(q) = q$, then we stop successfully.
(iii) Otherwise q lies in a c_e-cycle of size l. Apply p_e to x to search for some non-zero $i \leq 2^{e+x+1}$ such that $p^i_e(x)$ lies in a c_e-cycle of size $3e+1$. We call this element y_0. If y_0 is not found, then we stop *unsuccessfully*. If we reach the root q before 2^{e+x+1} applications of p_e, then we stop *successfully*.
(iv) Assume that y_0 is found. Compute the values $y_j = p^{2^{e+1}}(y_{j-1})$ for $1 \leq j \leq x+1$. If we reach the root q before finding y_{x+1}, then we stop successfully. If there is at least one y_j such that $1 \leq j \leq x+1$ and it lies in a c_e-cycle of size $3e+1$, then we stop successfully. Otherwise, we stop unsuccessfully.

The procedure always stops, either successfully or unsuccessfully. Informally speaking, if the described procedure stops unsuccessfully, then we can ensure that $\mathcal{C}_e$ is not a copy of $\mathcal{T}$. For each e, the function $(q,l,x) \mapsto u_{q,l,e}(x)$ is primitive recursive, since all searches are bounded. On the other hand, the function $(e,q,l,x) \mapsto u_{q,l,e}(x)$ is not primitive recursive. Nevertheless, the graph

$$\{(e,q,l,x,z) : u_{q,l,e}(x) = z\}$$

is a primitive recursive set.

We define the function $\alpha(l, \langle e, m\rangle)$. The intended use of the parameters is as follows: The position of any element x from the structure $\mathcal{T}$ is uniquely determined by the root $r_T(x)$ and the island coordinates (i,j) of x. Thus, the output of $\alpha(l, \langle e, m\rangle)$ will be the length of the c_T-cycle of the element x such that the root $r_T(x)$ lies in a cycle of size l and x has island coordinates $(\langle e, m\rangle, 0)$.

The function α is defined by primitive recursion on m. Suppose that l and e are fixed.

- We set $\alpha(l, \langle e, m\rangle) = 3e + 1$ for all m such that $\mathcal{C}_e[m]$ does not yet have a c_e-cycle C of size l and an element $q \in C$ with $r_e(q) = q$.
- Suppose that m_0 is the first number such that $\mathcal{C}_e[m_0]$ has a c_e-cycle of size l with root q. Note that if there is no such m_0, then simply $\alpha(l, \langle e, m\rangle) = 3e + 1$ for all m.
- For every $m_0 \leq m \leq u_{q,l,e}(0)$ we set $\alpha(l, \langle e, m\rangle) = 3e+1$. If (the procedure) $u_{q,l,e}(0)$ stops unsuccessfully, then define $\alpha(l, \langle e, m\rangle) = 3e + 1$ for all $m > u_{q,l,e}(0)$.
- Otherwise, $u_{q,l,e}(0)$ stops successfully. Then for m with $u_{q,l,e}(0) < m \leq u_{q,l,e}(1)$ we propagate the pattern $[3e+1, 3e+2]$. More formally, we define:

$$\alpha(l, \langle e, u_{q,l,e}(0) + 2j + 1\rangle) := 3e + 1,$$
$$\alpha(l, \langle e, u_{q,l,e}(0) + 2j + 2\rangle) := 3e + 2$$

  for appropriate j. In other words, we alternate between the cycles of size $3e + 1$ and $3e + 2$.

  In order to complete the pattern correctly, we assume that $(u_{q,l,e}(1) - u_{q,l,e}(0))$ is an even number. If $u_{q,l,e}(1)$ stops unsuccessfully, then we iterate the pattern $[3e + 1, 3e + 2]$ forever, i.e. for all $m > u_{q,l,e}(1)$.
- Otherwise, $u_{q,l,e}(1)$ stops successfully. In general, if some $u_{q,l,e}(k)$ stops unsuccessfully, then we end up repeating the pattern

$$[3e + 1, \underbrace{3e + 2, \ldots, 3e + 2}_{k \text{ times}}] \tag{3.1}$$

  forever. If every $u_{q,l,e}(k)$ stops successfully, then the pattern (3.1) is propagated for $u_{q,l,e}(k - 1) < m \leq u_{q,l,e}(k)$. In order to do this correctly, we assume that $(u_{q,l,e}(k) - u_{q,l,e}(k - 1))$ is divisible by $k + 1$.

Since the approximation $\mathcal{C}_e[s]$ is evaluated in bounded many steps and the graph of u is primitive recursive, the function α is also primitive recursive: Indeed, in the definition of α, we only need to decide whether $u_{q,l,e}(k)$ is equal to m or not.

The *working patterns* in our construction work as follows:

In Phase 1: Recall that w is the witness of an active strategy. Suppose that l is the length of the c_T-cycle of w. Let m be the least number such that $s^m_T(w)$ is still undefined. Then the length of the c-cycle of the newly added element $s^m_T(w)$ is defined as $\alpha(l, m)$. Define p and r for the new cycle appropriately.

In Phase 2: We grow $\mathcal{B}$ as follows. Suppose that v is the latest element with $r_B(v) = v$ which was added to $\mathcal{B}$ (i.e. v is the latest root inside $\mathcal{B}[s]$). Let l be the length of the c_B-cycle of v. Find the least m such that $s^m_B(v)$ is still undefined. Then define $s^m_B(v)$ as the next element a from the domain, and set the length of the c_B-cycle of a equal to $\alpha(l, m)$. Define p and r appropriately. Copy the newly added cycle into the structure $\mathcal{T}$.

Verification. At each stage of the construction, all evaluated functions are time-bounded. Therefore, the structures $\mathcal{B}$ and $\mathcal{T}$ are punctual.

Lemma 3.1. *Every requirement P_e is satisfied.*

Proof. First, we show the following: If there is a stage s_0 at which the P_e-strategy is active, then P_e is satisfied.

Let w be the P_e-witness, and l be the size of the c_T-cycle of w. Consider the least stage $s_1 \geq s_0$ such that $h_e(w)[s_1]\downarrow \in \mathcal{B}[s_1]$. The construction guarantees that $\mathcal{B}[s_1]$ contains no cycles of size l. Therefore, h_e is not an isomorphism from $\mathcal{T}$ onto $\mathcal{B}$, and P_e will be satisfied at stage s_1.

Now suppose that $s^* > e$ is the least stage such that:

- every P_i, $i < e$, is in state S4 (i.e. finished) at the beginning of the stage s^*; and
- the calculation of the set F of forbidden tags for P_e can be finished in $(s^* + 1)$ steps.

If P_e has never been active before the stage s^*, then it will become active at s^*. Thus, P_e will be satisfied. □

Now we can deduce that the structures $\mathcal{B}$ and $\mathcal{T}$ are isomorphic: This is ensured by copying $\mathcal{B}$ from $\mathcal{T}$ at Phase 1 (this happens infinitely often by Lemma 3.1) and by copying $\mathcal{T}$ from $\mathcal{B}$ at Phase 2.

Lemma 3.2. *Every requirement R_e is satisfied.*

Proof. Suppose that $\mathcal{C}_e$ is isomorphic to $\mathcal{T}$ and G is the unique isomorphism from $\mathcal{C}_e$ onto $\mathcal{T}$. First, we show that for every $x \in \mathcal{C}_e$, the length l_x of the c_e-cycle of the root $r_e(x)$ can be calculated primitively recursively.

Suppose that l_x is a tag from the set $L(P_i)$ for some i. Assume that $x \leq i - e$. Then the procedure $PROC^0_{i,e,l_x}$ must output "forbid l_x." Hence, Phase 0.3 of the construction guarantees that the structure $\mathcal{C}_e$ contains a c-cycle of size $l^* \in L(P_i)$ such that $l^* \in F$, i.e. l^* is a forbidden tag. Thus, $\mathcal{T}$ does not contain cycles of size l^*, and $\mathcal{T}$ is not isomorphic to $\mathcal{C}_e$. We obtained a contradiction.

Therefore, we have $x > i - e$ and $i < e + x$. This implies that the island tag l_x can be found by evaluating the elements

$$r_e(x), c_e(r_e(x)), c_e^2(r_e(x)), \ldots, c_e^{\max L(P_{e+x})}(r_e(x)),$$

where the function $k \mapsto \max L(P_k)$ is primitive recursive. Hence, the function $x \mapsto l_x$ is also primitive recursive.

Now we describe how to compute the image $G(q)$ of a root q from $\mathcal{C}_e$. First, find the corresponding island tag l_q and the index i_q such that $l_q \in L(i_q)$. Suppose that t^* is the number of steps needed to see that q belongs to a c_e-cycle of size l_q inside $\mathcal{C}_e$. W.l.o.g., one may assume that $t^* > e + q > i_q$. We argue that the element $G(q)$ must already belong to $\mathcal{T}[t^*]$.

Assume that $G(q) \notin \mathcal{T}[t^*]$. Consider Phase 0.2 of the stage t^*. Since $G(q) \notin \mathcal{T}[t^*]$, at the beginning of this phase, the strategy P_{i_q} must be in state S2: indeed, this is ensured by the description of Phase 0. Moreover, the strategy P_{i_q} does not have labels $\boxed{forb(e, l')}$.

If $e > i_q$, then w.l.o.g., one may assume that t^* is strictly greater than the number of steps needed to compute the set of forbidden tags F for the strategy P_{i_q}. Thus, here we need to consider only the case when $e \leq i_q$. Then the procedure $PROC_{i_q,e,l_q}$ finishes in $t^* + 1$ steps and outputs "forbid l_q." Therefore, P_{i_q} obtains label $\boxed{forb(e, l_q)}$. This implies that $\mathcal{T}$ cannot be isomorphic to $\mathcal{C}_e$ — a contradiction.

Therefore, $G(q)$ belongs to $\mathcal{T}[t^*]$, and we can compute the image $G(q)$ by proceeding with t^* stages of our construction. Hence, $G(q)$ is calculated in a primitive recursive way.

Finally, for an arbitrary natural number x, we show how to compute its image $G(x)$ in a primitive recursive way. First, we calculate the values $q = r_e(x)$ and $l = l_q$.

It is not hard to prove that the following two conditions are equivalent:

- x belongs to the c_e-cycle of its root $r_e(x)$;
- $p_e(s_e(x)) = r_e(x)$.

Hence, if $p_e(s_e(x)) = q$, then we can find the number $k \leq l$ with $x = c_e^k(q)$, and set $G(x) = c_T^k(G(q))$.

Suppose that $p_e(s_e(x)) \neq q$. Let (m, k) be the island coordinates of the element x. In order to compute $G(x)$, it is sufficient to describe a fast procedure for calculating the first island coordinate m: Indeed, if we know that $m = \langle i, j\rangle$ for some i and j, then the size of the c_e-cycle of x is equal to either $3i + 1$ or $3i + 2$. Hence, we have $k \leq 3i + 2$, and after finding $G(q)$, one can determine the image $G(x)$ in a straightforward way.

Recall that for a fixed e, the function $u_{q,l,e}(z)$ is primitive recursive. We show that $u_{q,l,e}(z)$ stops successfully for all z. Assume that there is the least z such that $u_{q,l,e}(z)$ stops unsuccessfully. Then for all elements from $\mathcal{T}$ with island coordinates $\langle e, j\rangle$, where $j > u_{q,l,e}(z-1)$, the R_e-strategy will iterate the pattern

$$[3e+1, \underbrace{3e+2, 3e+2, \ldots, 3e+2}_{z \text{ times}}].$$

This implies that z is the maximal possible number such that there are z many consecutive j-s such that $s_e^{\langle e,j\rangle}(q)$ has a c_e-cycle of size $3e + 2$.

The procedure $u_{q,l,e}(z)$ can stop unsuccessfully because of two reasons:

(1) either y_0 is not found, or
(2) each of the elements $y_1, y_2, \ldots, y_{z+1}$ belongs to a c_e-cycle of size $3e + 2$.

Each of these cases gives a contradiction. Therefore, $u_{q,l,e}(z)$ stops successfully for all z.

We claim that the first island coordinate m of the element x is at most $u_{q,l,e}(x)$. Assume that $m > u_{q,l,e}(x)$. Since $u_{q,l,e}(x)$ stops successfully, the procedure $u_{q,l,e}(x)$ will find a sequence

$$y_{z+1} = s_e^{\langle e,j\rangle}(q),\ y_z = s_e^{\langle e,j+1\rangle}(q),\ y_{z-1} = s_e^{\langle e,j+2\rangle}(q),\ \ldots,\ y_0 = s_e^{\langle e,j+z+1\rangle}(q),$$

which lies close to x and satisfies the following: $z \leq x$; each of y_1 and y_{z+1} belong to a c_e-cycle of size $3e + 1$, and each y_u, $1 \leq u \leq z$, has a c_e-cycle of size $3e + 2$. This contradicts the following: after $u_{q,l,e}(x)$ our construction propagates patterns

$$[3e+1, \underbrace{3e+2, 3e+2, \ldots, 3e+2}_{v \text{ times}}],$$

where $v > x$. Hence, $m \leq u_{q,l,e}(x)$, and the island coordinates of x can be computed in a primitive recursive way. Lemma 3.2 is proved. □

Theorem 1.4 is proved. □

4. Proof of Theorem 1.7

Proof idea. This time the structure will consist of two ω-chains, one generated by s and the other one by $\widehat{s}$. The chain generated by s will be similar to the one used in the previous theorem but without the extra "islands". It will be playing the role of the coordinate axis in the structure, with patterns of loops playing the role of the coordinates. It will be also used to press only the honesty generated punctual copies of the structure rather than all of its punctual copies.

The other ω-chain which is generated by $\widehat{s}$ will have no loops attached to it. We will also have another unary function f mapping points of the s-chain to points of the $\widehat{s}$-chain. See Fig. 2 below.

Since f^{-1} does not have to be primitive recursive, we can use the $\widehat{s}$-chain to diagonalise against isomorphisms to other punctual copies. However, in a honestly generated copy f^{-1} is primitive recursive, and therefore we are able to quickly find its s-chain coordinate. This feature will also allow us to press the honestly generated structures using the s-chain.

Proof. The language of our structure $\mathcal{A}$ consists of the following symbols:

- two constants o and $\widehat{o}$, and
- unary functions s, c, $\widehat{s}$, $\widehat{r}$, and f.

The (isomorphism type of the) structure $\mathcal{A}$ satisfies the following:

(a) The domain of $\mathcal{A}$ consists of two infinite disjoint parts D and $\widehat{D}$. We put $o \in D$ and $\widehat{o} \in \widehat{D}$. For any $x \in D$, we have $\widehat{r}(x) = o$. For $y \in \widehat{D}$, set $\widehat{r}(y) = \widehat{o}$. Note that, in particular, we will be able to quickly compute whether a given z belongs to D or $\widehat{D}$.

(b) If $x \in D$, then we define $\widehat{s}(x) = x$. If $y \in \widehat{D}$, then set $s(y) = c(y) = y$ and $f(y) = o$.

(c) The set $\widehat{D}$ forms an ω-chain relative to the function $\widehat{s}$: more formally,

$$\widehat{D} = \{\widehat{s}^n(\widehat{o}) : n \in \omega\},$$

and $\widehat{s} \restriction \widehat{D}$ is injective.

(d) The substructure $(D; s, c)$ satisfies the same properties as the structure $\tilde{\mathcal{A}} := (dom(\mathcal{A}); s, c)$ from the proof of Theorem 1.4. Informally speaking, $(D; s, c)$ is a version of the structure from Theorem 1.4, but with the predecessor and the root functions omitted.

(e) If $x = s^n(o)$ for some n, then $f(x) \in \widehat{D}$. Furthermore, if $m \neq n$, then $f(s^n(o)) \neq f(s^m(o))$. We assume that $f(o) = \widehat{o}$.

(f) If $x \in D$ and $x \neq s^n(o)$ for all n, then $f(x) = o$. We emphasize that $range(f) \subseteq \widehat{D} \cup \{o\}$.

It is not difficult to show that these properties ensure that the structure $\mathcal{A}$ is rigid and 1-generated (see Figure 2 below). As in Theorem 1.4, by $\mathcal{B}$ we denote the canonical presentation of $\mathcal{A}$.

Consider a language L_g which is obtained from the language of the structure $\mathcal{A}$ by adding a new unary function g. The intuition behind g is the following: We want to treat g as the "inverse" of the function f. More formally, we define a structure $\mathcal{A}_g$ in the language L_g as follows:

(1) the $L(\mathcal{A})$-reduct of $\mathcal{A}_g$ is equal to $\mathcal{A}$, and

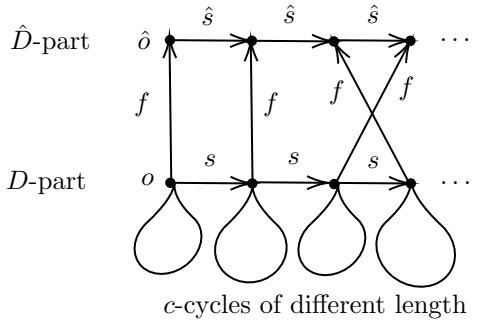

FIGURE 2. The structure $\mathcal{A}$ from Theorem 1.7

(2) we set

$$g(x) = \begin{cases} s^n(o), & \text{if } x \in \widehat{D} \text{ and } f(s^n(o)) = x, \\ \widehat{o}, & \text{otherwise.} \end{cases}$$

Note that the properties of $\mathcal{A}$ imply that the function g is well-defined (indeed, there is at most one number n with $f(s^n(o)) = x$). If $\mathcal{U}$ is a copy of $\mathcal{A}$, then the structure $\mathcal{U}_g$ (in the language L_g) is defined in a natural way.

Remark 4.1. If $\mathcal{U}$ is an honestly generated copy of $\mathcal{A}$, then the structure $\mathcal{U}_g$ is fpr.

As in Theorem 1.4, fix a uniformly computable list of all primitive recursive unary functions $\{h_e\}_{e\in\omega}$. Also choose a uniformly computable list of all fpr structures in the language L_g:

$$\mathcal{C}_n = (\omega; o_n, \widehat{o}_n, s_n, c_n, \widehat{s}_n, \widehat{r}_n, f_n, g_n),\ n \in \omega.$$

We will build an fpr copy $\mathcal{T}$ of $\mathcal{B}$, and satisfy the following series of requirements:

P_e: h_e is not an isomorphism from $\mathcal{T}$ onto $\mathcal{B}$.
Q_n: If $\mathcal{C}_n \cong \mathcal{B}_g$, then there is a primitive recursive isomorphism from $\mathcal{C}_n$ onto $\mathcal{B}_g$.

Furthermore, the construction will ensure that the structure $\mathcal{B}$ is honestly generated (see the verification). This fact and the requirements above are enough to prove the theorem:

(1) The P_e-requirements guarantee that $|\mathbf{FPR}(\mathcal{A})| > 1$.
(2) If $\mathcal{U}$ is an honestly generated copy of $\mathcal{A}$, then there is an index n such that $\mathcal{C}_n = \mathcal{U}_g$. The Q_n-requirement implies that the unique isomorphism F from $\mathcal{U}$ onto $\mathcal{B}$ is primitive recursive. Since $\mathcal{B}$ is the canonical copy of $\mathcal{A}$, the (unique) isomorphism $G = F^{-1}$ from $\mathcal{B}$ onto $\mathcal{U}$ is also primitive recursive.

An overview.

As in Theorem 1.4, at a stage s, the finite structure $\mathcal{T}[s]$ will have a part that copies $\mathcal{B}$ and an *island* part $I[s]$. The island part contains *no* elements from D, i.e. for every $x \in I[s]$, we have $\widehat{r}_T(x) = \widehat{o}$.

Since we do not use the predecessor and the root functions as before (see Theorem 1.4), the machinery of island tags is completely omitted. Every strategy P_e can be in one of the following states:

(S1) unstarted;
(S2) active;
(S3) finished.

While P_e is active, it is building its own P_e-island. This island is just a finite piece of a $\mathbb{Z}$-chain, with respect to the function $\widehat{s}_T$. Since $\mathcal{T}$ *does not have to be honestly generated*, we exploit this fact and *do not connect* the P_e-island I to the non-island part of $\mathcal{T}[s]$: in particular, this includes not putting elements from I into $range(f_T)[s]$. Only when P_e is finished, we will put (some) elements from I into $range(f_T)$.

In order to ensure that $\mathcal{B}$ is honestly generated, we will guarantee that:

(a) If $\xi \in \{s, c, \widehat{s}\}$, then for any $x \in \omega$, the set $\xi_B^{-1}(x)$ is finite. Moreover, given x, one can promptly compute the Gödel index of the finite set $\xi_B^{-1}(x)$.
(b) The set $range(f_B)$ is equal to $\widehat{D}(\mathcal{B})$ and given $y \in range(f_B)$, we can quickly find the unique x with $f_B(x) = y$.

The item (b) above can be obtained by a careful working with the function f_B: At the end of each stage s, we find all $y \in \widehat{D}(\mathcal{B})[s]$ such that $y \notin range(f_B)[s]$. For each such y, we attach a fresh c_B-cycle C_y to $\mathcal{B}$. Suppose that $x = s_B^n(o)$ lies in C_y. Then we set $f_B(x) := y$. After that, we extend the non-island part of $\mathcal{T}$ to match the current $\mathcal{B}$. This action guarantees that every $y \in \widehat{D}$ belongs to $range(f)$.

The strategy P_e in isolation. Pick the least unused element w_e and put it into $\widehat{D}(\mathcal{T})$. Declare w_e the P_e-*witness*. While the value $h_e(w_e)[t]$ is either undefined or does not belong to $\mathcal{B}[t]$, proceed with constructing the P_e-*island* inside $\mathcal{T}$ as follows.

Recall that the function $\widehat{s} \restriction \widehat{D}$ is injective, hence, for a number $z \in \widehat{D} - \{\widehat{o}\}$, one can consider its unique preimage $\widehat{s}^{-1}(z)$. Assume that at a stage t, both the values $\widehat{s}^k(w_e)$ and $\widehat{s}^{-k}(w_e)$ are already defined. Then we pick next elements y and z from the domain of $\mathcal{T}$, and set $\widehat{s}^{k+1}(w_e) = y$ and $\widehat{s}^{-k-1}(w_e) = z$. The other functions (from the language of $\mathcal{T}$) are defined in an appropriate way: e.g., $\widehat{s}(z) = \widehat{s}^{-k}(w_e)$, $s(z) = c(z) = z$, $\widehat{r}(z) = \widehat{o}$, and $f(z) = o$. While the P_e-island is growing, we do not put w_e into $range(f_T)$.

Suppose that t_0 is the first stage such that $h_e(w_e)[t_0]\downarrow \in \mathcal{B}[t_0]$. Then find the least even length $l_e > 0$ such that we have never used c-cycles of size l_e before. We attach the P_e-island to the main part of $\mathcal{T}[t_0]$ and extend $\mathcal{B}$ to an isomorphic copy of $\mathcal{T}$. More formally, proceed as follows:

(1) Let k be the least number such that $\widehat{s}_T^k(\widehat{o})$ is still undefined. Find the greatest number m such that $\widehat{s}_T^{-m}(w_e)$ is already defined. Set $\widehat{s}_T^k(\widehat{o}) := \widehat{s}_T^{-m}(w_e)$.
(2) Let n be the least number such that $s_T^n(o)$ is undefined. W.l.o.g., we may assume that n is even and $n \neq 0$. Then we form a fresh c_T-cycle C of size l_e inside $\mathcal{T}$, in such a way that $s_T^n(o) \in C$. We define $f_T(s_T^n(o)) := w_e$.
(3) Grow the main part of $\mathcal{B}$ by attaching the copy of the P_e-island and the copy of C, in a natural way.

After that, the requirement P_e is declared *satisfied*: Indeed, either the element $h_e(w_e)$ does not belong to $range(f_B)$, or its preimage $f_B^{-1}(h_e(w_e))$ lies in a c_B-cycle of length $l' \neq l_e$. Therefore, h_e cannot be an isomorphism from $\mathcal{T}$ onto $\mathcal{B}$.

The strategy Q_e in isolation. This is similar to Theorem 1.4, but presses only honestly generated structures.

First, consider the element $x = 0$ from $\mathcal{C}_e$. Calculate the value $\widehat{r}_e(x)$. If $\widehat{r}_e(x) \notin \{o_e, \widehat{o}_e\}$, then trivially $\mathcal{C}_e$ is not a copy of $\mathcal{B}_g$. Therefore, w.l.o.g. we may assume that $\widehat{r}_e(x) \in \{o_e, \widehat{o}_e\}$, and one can quickly determine whether x belongs to $D(\mathcal{C}_e)$ or to $\widehat{D}(\mathcal{C}_e)$.

Now we assume that $\mathcal{C}_e$ is a copy of $\mathcal{B}_g$, and we want to quickly compute the value $F(x)$, where F is the unique isomorphism from $\mathcal{C}_e$ onto $\mathcal{B}_g$. Note that one can promptly find a new value $x^{\#}$ which is defined as follows:

(1) If $x \in D(\mathcal{C}_e)$, then set $x^{\#} := x$.
(2) If $x \in \widehat{D}(\mathcal{C}_e)$, then define $x^{\#} := g_e(x)$. If $\mathcal{C}_e \cong \mathcal{B}_g$, then here x should belong to $range(f_e)$ (see the discussion at the end of the overview), and $f_e(x^{\#})$ must be equal to x.

The element $x^{\#}$ always belongs to $D(\mathcal{C}_e)$ (if it does not, then $\mathcal{C}_e \not\cong \mathcal{B}_g$). Furthermore, it is not hard to show the following: if one can promptly find the value $F(x^{\#})$, then it is also possible to quickly compute the desired $F(x)$. Therefore, for simplicity, we will assume that $x = x^{\#} \in D(\mathcal{C}_e)$.

Apply s_e to x at most 2^{e+2} times to find an element y_0 belonging to a c_e-cycle of size $4e+1$. When y_0 is found, compute $y_1 = s_e^{2^{e+2}}(y_0)$. Check whether y_1 belongs to a c_e-cycle of size $4e+1$. Suppose that this procedure takes u_0 steps. If the c_e-cycle of y_1 has size $4e+1$, then we say that the number u_0 has been calculated *successfully*.

While we are waiting for u_0 to be calculated, we do not delay the construction of $\mathcal{B}$ and $\mathcal{T}$:

- One by one, we add new c_B-cycles C into $\mathcal{B}$. Let n be the number such that $s_B^n(o) \in C$:
  - If n is even (or in other words, $n = \langle 0, m\rangle$ for some m), then the length l of C is equal to 2.
  - If $n = \langle e+1, m\rangle$ for some m, then we set $l := 4e+1$.

  Informally speaking, we iterate the pattern $4e+1, 4e+1, 4e+1, \ldots$ in appropriate coding places.
- For each cycle C from above, we put a fresh element z_C inside (the growing ω-chain) $\widehat{D}(\mathcal{B})$ and set $f_B(s_B^n(o)) := z_C$.
- We extend the non-island part of $\mathcal{T}$ to an isomorphic copy of current $\mathcal{B}$.

If u_0 is not defined or u_0 has been calculated unsuccessfully, then $\mathcal{C}_e \not\cong \mathcal{B}_g$ and we just continue putting $(4e+1)$-cycles. If u_0 is calculated successfully, we will ensure that $\mathcal{B}$ will not get new adjacent c_B-cycles of size $4e+1$.

Now assume that $x = 1$. Again, for simplicity, we deal with $x \in D(\mathcal{C}_e)$. Apply s_e to x at most 2^{e+3} times to find an element y_0 belonging to a c_e-cycle of size $4e+1$. When y_0 is found, compute $y_1 = s_e^{2^{e+2}}(y_0)$ and $y_2 = s^{2^{e+2}}(y_1)$. Check whether at least one of y_1 or y_2 belongs to a c_e-cycle of size $4e+1$. If it is the case, then the procedure is finished successfully. The number u_1 is defined as the number of steps needed to finish this procedure (either successfully or unsuccessfully).

While waiting for the procedure to finish, grow $\mathcal{B}$ and $\mathcal{T}$ as above, modulo the following important modification: We alternate between c_B-cycles of size $4e+1$ and $4e+3$ for $s_B^{\langle e+1,m\rangle}(o)$:

$$4e+1, 4e+3, 4e+1, 4e+3, \ldots.$$

In other words, we propagate the pattern $[4e+1, 4e+3]$.

Again, if u_1 is never calculated or calculated unsuccessully, then we continue the pattern $[4e+1, 4e+3]$, and $\mathcal{C}_e$ is not a copy of our structure. If u_1 is calculated successfully, then we switch to $x = 2$ and to the pattern

$$4e+1, 4e+3, 4e+3, 4e+1, 4e+3, 4e+3, \ldots.$$

Now we describe the general procedure. Consider $x = n > 1$. Apply s_e to x at most 2^{e+n+2} times to find y_0 which belongs to a c_e-cycle of size $4e+1$. When y_0 is found, compute $y_{i+1} = s_e^{2^{e+2}}(y_i)$, for $0 \leq i \leq n$. Check whether one of the elements $y_1, y_2, \ldots, y_{n+1}$ belongs to a c_e-cycle of size $4e+1$. If it is the case, then we say that the procedure finished *successfully*.

Let u_n be the number of steps needed to finish the procedure. While u_n is being computed, we propagate the pattern

$$[4e+1, \underbrace{4e+3, \ldots, 4e+3}_{n \text{ times}}]$$

in appropriate coding places. If u_n is calculated unsuccessfully or u_n is never computed, then we continue iterating the same pattern forever. If u_n is calculated successfully, then we start considering $x = n+1$ and switch to the next pattern

$$[4e+1, \underbrace{4e+3, \ldots, 4e+3}_{n+1 \text{ times}}].$$

The calculations of u_i, $i \in \omega$, will allow us to quickly compute the isomorphism $F \colon \mathcal{C}_e \to \mathcal{B}_g$.

Construction. Similar to Theorem 1.4, but instead of pressing more punctual structures at later stages we press more honestly generated structures. The actions of the diagonalisation P-requirements are finitary.

Verification. It should be clear from the description of P_e that the diagonalisation is always successful. The verification of the Q-strategies is similar to Theorem 1.4. In fact, we do not really need the full power of being honestly generated here. All we need is having g (which plays the role of f^{-1}) primitive recursive in the opponent's structure. Using g we can punctually compute the coordinates of any point in the $\widehat{s}$-chain of the opponent's structure and then promptly match it with the respective point in our structure. Otherwise, the verification of the Q-strategies is the same as in the verification of Theorem 1.4. □

References

[AKNS] M. Aschenbrenner, A. Khélif, E. Naziazeno, and T. Scanlon. The logical complexity of finitely generated commutative rings. *Int. Math. Res. Not.* doi: 10.1093/imrn/rny023.

[AT51] E. Artin and J. Tate. A note on finite ring extensions. *J. Math. Soc. Japan*, 3:74–77, 1951.

[BDKM] N. Bazhenov, R. Downey, I. Kalimullin, and A. Melnikov. Foundations of online structure theory. Preprint.

[BG00] Achim Blumensath and Erich Grädel. Automatic structures. In *15th Annual IEEE Symposium on Logic in Computer Science (Santa Barbara, CA, 2000)*, pages 51–62. IEEE Comput. Soc. Press, Los Alamitos, CA, 2000.

[BHTK$^+$] N. Bazhenov, M. Harrison-Trainor, I. Kalimullin, A. Melnikov, and K. M. Ng. Automatic and polynomial-time algebraic structures. Preprint.

[CR91] Douglas A. Cenzer and Jeffrey B. Remmel. Polynomial-time versus recursive models. *Ann. Pure Appl. Logic*, 54(1):17–58, 1991.

[CR98] D. Cenzer and J. B. Remmel. Complexity theoretic model theory and algebra. In Yu. L. Ershov, S. S. Goncharov, A. Nerode, and J. B. Remmel, editors, *Handbook of recursive mathematics, Vol. 1*, volume 138 of *Stud. Logic Found. Math.*, pages 381–513. North-Holland, Amsterdam, 1998.

[DF13] Rodney G. Downey and Michael R. Fellows. *Fundamentals of parameterized complexity*. Texts in Computer Science. Springer, London, 2013.

[DHTK$^+$] R. Downey, M. Harrison-Trainor, I. Kalimullin, A. Melnikov, and D. Turetsky. Graphs are not universal for online computablility. Preprint.

[Ers12] M. Ershov. Golod–Shafarevich groups: A survey. *Int. J. Algebra Comput.*, 22(5):1230001, 2012.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and intractability*. W. H. Freeman and Co., San Francisco, Calif., 1979. A guide to the theory of NP-completeness, A Series of Books in the Mathematical Sciences.

[Gol64] E. S. Golod. On nil-algebras and finitely approximable p-groups. *Izv. Akad. Nauk SSSR Ser. Mat.*, 28(2):273–276, 1964.

[Gro81] M. Gromov. Groups of polynomial growth and expanding maps. *Publications Mathématiques de L'Institut des Hautes Études Scientifiques*, 53(1):53–78, 1981.

[Hig61] G. Higman. Subgroups of finitely presented groups. *Proc. Roy. Soc. Ser. A*, 262:455–475, 1961.

[Kie81] H. A. Kierstead. An effective version of Dilworth's theorem. *Trans. Am. Math. Soc.*, 268:63–77, 1981.

[Kie98a] H. A. Kierstead. On line coloring k-colorable graphs. *Israel J. Math.*, 105(1):93–104, 1998.

[Kie98b] H. A. Kierstead. Recursive and on-line graph coloring. In Yu. L. Ershov, S. S. Goncharov, A. Nerode, and J. B. Remmel, editors, *Handbook of recursive mathematics, Vol. 2*, volume 139 of *Stud. Logic Found. Math.*, pages 1233–1269. North-Holland, Amsterdam, 1998.

[KM10] Bakhadyr Khoussainov and Mia Minnes. Three lectures on automatic structures. In *Logic Colloquium 2007*, volume 35 of *Lect. Notes Log.*, pages 132–176. Assoc. Symbol. Logic, La Jolla, CA, 2010.

[KMN17a] I. Sh. Kalimullin, A. G. Melnikov, and K. M. Ng. The diversity of categoricity without delay. *Algebra Logic*, 56(2):171–177, 2017.

[KMN17b] Iskander Kalimullin, Alexander Melnikov, and Keng Meng Ng. Algebraic structures computable without delay. *Theoret. Comput. Sci.*, 674:73–98, 2017.

[KN95] Bakhadyr Khoussainov and Anil Nerode. Automatic presentations of structures. In *Logic and Computational Complexity (Indianapolis, IN, 1994)*, volume 960 of *Lecture Notes in Comput. Sci.*, pages 367–392. Springer, Berlin, 1995.

[KPT94] H. A. Kierstead, S. G. Penrice, and W. T. Trotter Jr. On-line coloring and recursive graph theory. *SIAM J. Discrete Math.*, 7:72–89, 1994.

[Lew67] J. Lewin. Subrings of finite index in finitely generated rings. *J. Algebra*, 5(1):84–88, 1967.

[LS01] Roger C. Lyndon and Paul E. Schupp. *Combinatorial group theory*. Classics in Mathematics. Springer-Verlag, Berlin, 2001. Reprint of the 1977 edition.

[LST89] L. Lovász, M. Saks, and W. T. Trotter Jr. An on-line graph coloring algorithm with sublinear performance ratio. *Discrete Math.*, 75:319–325, 1989.

[Mel17] Alexander G. Melnikov. Eliminating unbounded search in computable algebra. In *Unveiling dynamics and complexity*, volume 10307 of *Lecture Notes in Comput. Sci.*, pages 77–87. Springer, Cham, 2017.

[MN] A. G. Melnikov and K. M. Ng. The back-and-forth method and computability without delay. Preprint.

[NA68] P. S. Novikov and S. I. Adjan. Infinite periodic groups. I. *Math. USSR Izv.*, 2(1):209–236, 1968.

[Nos83] G. A. Noskov. Elementary theory of a finitely generated commutative ring. *Math. Notes*, 33(1):12–15, 1983.

[Rem86] J. B. Remmel. Graph colorings and recursively bounded Π^0_1-classes. *Ann. Pure Appl. Logic*, 32:185–194, 1986.

[Rog87] H. Rogers. *Theory of recursive functions and effective computability*. MIT Press, Cambridge, MA, second edition, 1987.

Sobolev Institute of Mathematics
E-mail address: bazhenov@math.nsc.ru

Kazan Federal University
E-mail address: ikalimul@gmail.com

Massey University & Kazan Federal University
E-mail address: alexander.g.melnikov@gmail.com

Nanyang Technological University
E-mail address: selwyn.km.ng@gmail.com