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 PR-degrees (to be defined). We confirm a natural conjecture by showing that the PR-degrees of a finitely generated structure must be dense. Remarkably, we show that PR-degrees of an f.g. structure do not have to be upwards dense. As an application of our techniques, we refute a natural conjecture about honestly generated structures (to be stated).

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

1. Introduction and results

Modern online algorithms are typically 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 on online infinite combinatorics [Kie81, Kie98, KPT94, LST89, Rem86] and, rather independently, polynomial time and automatic infinite algebraic structures [KN95, BG00, KM10, CR91, CR98]. Nonetheless, until the recent work of Kalimullin, Melnikov and Ng [KMN17b] there had been no systematic theoretical framework that would unite these two fairly independent subjects. The present article contributes to this recently suggested framework which aims to develop a general theory of online computation on potentially infinite data.

Of course, one naturally seeks to understand the truly efficient online algorithms. Kalimullin, Melnikov and Ng [KMN17b] 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 every search and loop. The effects of this seemingly relaxed restriction can be rather significant. This paper contributes to the general program that

2010 *Mathematics Subject Classification.* Primary 03D45, 03C57. Secondary 03D75, 03D80.

The work of the first two authors was supported by RSF grant no. 18-11-00028. Also the second author was funded by Russian government as a federal professor in mathematics.The third author was partially supported by Marsden Fund of New Zealand. The last author is partially supported by the grant MOE2015-T2-2-055.

systematically investigates these effects in computable algebra and combinatorics; see [KMN17b, Mel17, BDKM19, DHTK$^+$, KMN17a, MN]. See survey [BDKM19] for the foundations of the surprisingly rich emerging theory of primitive recursive ("punctual") structures, and see also Alaev [Ala17, Ala18b] for an alternative and independent approach. We note that the theory of punctual structures has recently found an unexpected application in the study of automatic structures; see [BHTK$^+$] for a solution of a long-standing open problem of Khoussainov and Nerode (e.g., Question 4.9 in [KN08]) on the classification of automata-presentable structures.

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 a finitely generated structure. 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. 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].

Notice, however, that the observation above uses a rather general notion of an algorithmically effective process, namely a Turing computable (or general recursive) process. For suppose we are given some algorithmic description $\mathcal{C}$ of $(\mathbb{Z}, +)$, 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 $c^m = x$, and then we can define $x \to m$. 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 therefore natural to ask what exactly happens when we forbid unbounded search. What used to be trivial from the perspective of general recursion theory now will have some non-trivial and often counterintuitive *subrecursive* content. 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 (at least) satisfy the following general definition.

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

We informally call fully primitive recursive structures *punctually computable* or simply *punctual*. Although the definition above is not restricted to finite languages,

we intend to keep our languages finite; in particular, will never encounter infinite languages in the present article[1].

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 $\equiv_{pr}$ and the degree structure on the classes which will be denoted $\mathbf{PR}(\mathcal{A})$.

What does $\mathbf{PR}(\mathcal{A})$ reflect? If $\mathcal{C} \leq_{pr} \mathcal{B}$ then, in a way, $\mathcal{B}$ has more subrecursive content than $\mathcal{C}$ does, in the sense 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 PR-degrees of the dense linear order, the random graph, and the universal abelian p-group are pairwise non-isomorphic; see [MN]. From the perspective of general Turing computability, the back-and-forth proofs of uniqueness for these standard homogeneous structures are equally effective. A shallow analysis suggests that all three back-and-forth procedures have exactly one potentially unbounded delay. Nonetheless, using a highly sensitive primitive recursive invariant – namely, the punctual degrees – we can detect the subtle differences in these seemingly identical back-and-forth proofs.

In the case of finitely generated structures the situation is somewhat similar to that for the rationals or the random graph. From the perspective of general Turing computability, any two computable presentations of a finitely generated structure are computably isomorphic. In other words, all finitely generated structures are relatively computably categorical [EG00, AK00]. Computable structure theory is usually concerned with structures which are not relatively computably categorical; in particular, not finitely generated.

From the point of view of *punctual* structure theory, finitely generated structures are no longer tame. But even we forbid unbounded search, the procedure of building an isomorphism on x seems to be exactly the same for all finitely generated structures. Namely, we search for a term over the generators which is equal to a

[1]It is also natural to ask why we did not choose the domain to be an arbitrary primitive recursive subset of $\mathbb{N}$. In the primitive recursive world this will give a provably non-equivalent notion of a primitive recursive structure [CR98, CR91]. The reason is that we can delay elements from appearing in the domain of such a structure; see [KMN17b] for examples that include undirected graphs. As has been observed by Alaev, if a primitive recursive structure is not locally finite or is punctually 1-decidable, then it can be primitively recursively isomorphically mapped onto its fully primitive recursive presentation ([Ala18a]). For instance, all structures considered in this paper are not locally finite, and therefore having the domain equal to $\mathbb{N}$ is a mere convenience for the most part. We strongly conjecture that all our results in the present paper hold for primitive recursive structures as well.

given element x. Nonetheless, there exist finitely generated infinite $\mathcal{A}$ and $\mathcal{B}$ with $\mathbf{PR}(\mathcal{A}) \not\cong \mathbf{PR}(\mathcal{B})$; see [KMN17b].

However, the property of being finitely generated seems to put a significant restriction on the punctual degrees of a structure. The underlying motivation here is to find a nontrivial link between the purely *algebraic* property "A is finitely generated" and its *algorithmic* invariant $\mathbf{PR}(A)$. This subject naturally splits into two main themes. In the first theme, one seeks to formulate a general enough property of PR-degrees of $\mathcal{A}$ which is implied by $\mathcal{A}$ being finitely generated. The second theme is concerned with finding finitely generated structures whose PR-degrees have non-trivial and counterintuitive features, e.g., refuting natural conjectures. While the two themes clearly complement each other both technically and methodologically, the second theme is of some special interest to a computable structure theorist because finitely generated structures are often completely overlooked in (Turing) computable structure theory.

The paper at hand contributes to both themes, but the most technically challenging Theorem 1.4 clearly belongs to the second theme. While we believe that the "monstrous" counterexample from Theorem 1.4 is of significant independent interest, the *novel techniques* developed to construct it have already found applications which contribute to the first "structural" theme. More specifically, in the satellite paper [KMZ] Kalimullin, Melnikov and Zubkov have used similar strategies and intuition to show that, for *any* finitely generated rigid $\mathcal{A}$ with $|\mathbf{PR}(\mathcal{A})| > 1$, a countable lattice $\mathcal{L}$ is embeddable into $\mathbf{PR}(\mathcal{A})$ iff $\mathcal{L}$ is distributive. Without a doubt this is a structural result which is yet to be fully understood. We will discuss one further recent application in due course. Now to the results.

1.2. **Results.** There are several basic properties of punctual degrees that are shared among all finitely generated (f.g.) structures. Clearly, each such structure has the least punctual degree which is the degree of the naturally generated copy of the structure (i.e., every element at every stage is a term of generators). We will also show that, for a f.g. $\mathcal{A}$, $|\mathbf{PR}(\mathcal{A})| = 1$ is equivalent to saying that every two punctual copies of $\mathcal{A}$ are isomorphic via a primitive recursive f having primitive recursive inverse; see Fact 2.1.

Harrison-Trainor, Melnikov and Turetsky have announced that there exists a punctual structure $\mathcal{B}$ for which $\mathbf{PR}(\mathcal{B})$ is not dense. The first main result of the paper says that in the case of finitely generated structures such pathological examples do not exist.

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

Kalimullin, Melnikov, and Ng [KMN17b] constructed a finitely generated punctual structure $\mathcal{A}$ with the property $|\mathbf{PR}(\mathcal{A})| = 1$, and therefore the punctual degrees of a finitely generated structure do not have to be upward dense. However, all natural examples of finitely generated structures seem to have the punctual degrees upward dense whenever $|\mathbf{PR}(\mathcal{A})| > 1$. Rather unexpectedly, this natural hypothesis fails in the strong sense below.

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

In some way, Theorem 1.4 should be compared with Goncharov's celebrated example of a structure having computable dimension two (e.g., [Gon81]), in the

sense that both results are counterintuitive and require significant novel technical insights. Since a finitely generated structure always has the least PR-degree, our first main result Theorem 1.3 excludes the possibility of $|\mathbf{PR}(\mathcal{A})| = 2$ for a finitely generated structure $\mathcal{A}$. The structure from Theorem 1.4 is thus the closest one can get to punctual dimension two with a finitely generated structure.

We have already mentioned above one (indirect) application of Theorem 1.3. We also note that Melnikov and Ng have recently announced that there is a (clearly, not f.g.) structure C with $|\mathbf{PR}(C)| = 2$. The methods that they used are similar to those in the proof of Theorem 1.4 but have virtually nothing in common with the above-mentioned proof of Goncharov.

The proof of Theorem 1.4 is combinatorially quite involved. It extends a strategy introduced in [KMN17b] and blends it with a new technique. While we put much effort into semi-formal explanation of the argument, a rather solid background in computable structure theory is perhaps necessary to understand the proof in full depth.

The high combinatorial complexity of our proof of Theorem 1.4 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 $\mathbf{PR}(\mathcal{A})$ is upward dense.*

We finish the paper with an application of the technique developed in the proof of Theorem 1.4. But first, we give some intuition.

In computable structure theory, adding prime root extraction to the language of an abelian group [Khi98] or assuming a factorisation algorithm in a field [Rab60] have profound effects to the global and local computability-theoretic properties of the respective structure. For example, adding p-root extraction may effect computable categoricity of an abelian p-group. However, even in Turing computability, all these augmentations of signatures by Skolem-like functions (to be made precise in Def. 1.6 below) make little difference when a structure is finitely generated. In particular, a finitely generated structure is already relatively computably categorical, and – from the perspective of Turing computable structures – it cannot be made much better than that. To clarify what we mean by a Skolem-like function, consider the definition below which is a bit weaker than punctual 1-decidability [KMN17b].

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 following conjecture is supported by many natural examples as well as by the pathological structure from Theorem 1.4.

Hypothesis: Suppose $|\mathbf{PR}(\mathcal{A})| = \infty$ for a finitely generated $\mathcal{A}$, then $\mathcal{A}$ also has infinitely many honestly generated copies, up to $\equiv_{pr}$.

Indeed, adding the predecessor unary function P into the language of (ω, S) will not result in $|\mathbf{PR}(\omega, S, P)| = 1$. Adding Skolem function for division or "$-$" (or both) into the language of finitely generated abelian groups does not seem to have

any significant effect on their PR-degrees either. Even if we could quickly find the elements that generate a given element via a given term (if they exist), it would not give us any extra power. The intuition here is that there could be a general notion of a "distance" which measures the size of the smallest term that generates a given element. The ability to honestly generate (Def. 1.6) the subalgebra $\langle x \rangle$ of C should not help us to promptly find the term expressing x via the generators $\bar{c}$, thus giving us enough diagonalization witnesses. Nonetheless, we apply our techniques to refute this conjecture, in particular showing that, in general, there is no well-behaved notion of a "distance" which would be enough to prove the conjecture.

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

We suspect that the structure constructed in the theorem possesses certain peculiar features related to the speed of growth which could be of some interest to combinatorial algebraists. 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 this as an open problem.

We also leave open if being honestly generated can be replaced by punctual 1-decidability in the theorem above. See [BDKM19, Mel17, KMN17b, Ala18a, Bli19] 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 1.3

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.

Recall that a punctual structure $\mathcal{S}$ is *punctually categorical* if for any punctual $\mathcal{A}$ isomorphic to $\mathcal{S}$, there is an isomorphism $f\colon \mathcal{A} \to \mathcal{S}$ such that both f and f^{-1} are primitive recursive. 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 $\mathcal{B} \not\geq_{pr} \mathcal{A}$ 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 $|\mathbf{PR}(\mathcal{A})| = 1$ 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 $|\mathbf{PR}(\mathcal{A})| = 1$. 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 1.3.** Let $\mathcal{A}$ be a finitely generated punctual structure. We need to prove that $\mathbf{PR}(\mathcal{A})$ is dense.

If $|\mathbf{PR}(\mathcal{A})| = 1$ 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 fix a computable listing $(p_e)_{e\in\omega}$ of all unary primitive recursive functions. 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 $g : \mathcal{X} \to \mathcal{T}$. All constructed objects will be build inductively by stages. At each stage we will extend the previous finite part of the structure $\mathcal{X}_s$ to some larger portion of the structure. Since our objects should be primitive recursive, we cannot apply an unbounded search before extending the definition of $\mathcal{X}_s, f_s$ and g_s. For instance, we cannot hope to immediately compute a desired value of p_e or p_j for the corresponding strategies of P_e and R_j because the uniform enumeration of all primitive recursive functions is not uniformly primitive recursive. Thus, we will have to keep extending the definition of $\mathcal{X}_s, f_s$ and g_s and wait for p_e or p_j to converge.

The idea is to build $\mathcal{X}$ by copying either $\mathcal{B}$ or $\mathcal{T}$, and switch between letting $\mathcal{X}$ copy $\mathcal{B}$ and letting $\mathcal{X}$ copy $\mathcal{T}$. However, this would not be possible in general without $\mathcal{A}$ being finitely generated. Fix the generators $\bar{a}$ of $\mathcal{B}$. Recall that $\mathcal{B} <_{pr} \mathcal{T}$, and assume h witnesses this reduction. We let $\bar{a}''$ be the h-image of $\bar{a}$ 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 $\bar{a}'$ be the image of $\bar{a}$ in $\mathcal{X}$. For the benefit of the uninitiated, recall that the domain of each structure is ω, and we have to generate the term algebra of $\mathcal{X}_s$ and assign elements in the domain to elements of the term algebra in a primitive recursive way. By letting "$\mathcal{X}$ copy $\mathcal{B}$" we mean that we generate the term algebra for $\mathcal{B}_s$ and $\mathcal{X}_s$ up to the same number of steps and define f on these elements appropriately. If there are elements in $\mathcal{B}_s$ which are currently independent from $dom(f_s)$ then we introduce new elements into $\mathcal{X}_{s+1}$ for the images of these elements. Basically, we want to ensure that when copying $\mathcal{B}$ into $\mathcal{X}$, f_s is a finite isomorphism between the two finite substructures (and g_s will be an embedding between $\mathcal{X}_s$ and $\mathcal{T}_s$ and not necessarily surjective). A similar idea

holds when $\mathcal{X}$ is copying $\mathcal{T}$; in this case, g_s will be a finite isomorphism between $\mathcal{X}_s$ and $\mathcal{T}_s$ while f_s is a non-surjective embedding.

The strategy for P_e. In the construction, we will initially start by waiting until we see that $p_e(\bar{a}')$ generates $\bar{a}$ in $\mathcal{B}$. If this never happens then p_e cannot be an onto isomorphism, and thus the requirement will be met. (Of course we have to continue extending $\mathcal{X}_s$, f_s and g_s and attending to other requirements while waiting for P_e). Suppose we see that $p_e(\bar{a}')$ generates $\bar{a}$ 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 between P_e and the requirement currently in control of the construction)

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 copying $\mathcal{B}$ via f, and assume we want 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 $\mathcal{T}$. This will ensure that $h = g \circ f$ can be maintained.

In other words after the switch we start to add into $\mathcal{X}$ the images of elements of $\mathcal{T}$ which formerly could not be added because they were not yet enumerated into $h(\mathcal{B})$.

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. In other words, we keep growing $\mathcal{X}$ by generating only the term algebra of existing elements and we do not continue identifying new independent elements of $\mathcal{T}$ inside $\mathcal{X}$. (This is necessary for keeping $\mathcal{X}$ punctual, as we cannot delay extending $\mathcal{X}$ while waiting for $h(\mathcal{B})$ to catch up).

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, so this delay in extending $\mathcal{X}_s$ is justified. 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 $\mathcal{B}$. From now we can simply copy the structure $\mathcal{B}$ into $\mathcal{X}$ almost identically up to a finite permutation of elements.

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 would 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. For simplicity, first suppose we have to merely build $\mathcal{A}$ with exactly one punctual degree using the strategy from [KMN17b] (Proposition 4.2). Then we will extend this strategy to construct an $\mathcal{A}$ with the desired properties. We give only a very brief global outline of this strategy; see [KMN17b] for further details, and see survey [BDKM19] for a detailed informal outline of this strategy.

The structure $\mathcal{A}$ will consist of an ω-chain constructed using a unary function s:

$$o \to s(o) \to s^2(o) \to \cdots,$$

where each element of the form $x = s^k(o)$ is a part of an *m-cycle* (for some m):

$$x \to c(x) \to c^2(x) \to \cdots c^{m-1}(x) \to x.$$

All such m-cycles will be disjoint. Also, we fix a third unary function p that maps each point of the form $s^t(c^k(o))$ back to the fixed generator o. (We omit some trivial details of, e.g., how exactly we define s on the new points forming the c-cycles, etc.) If we choose the sizes m for different k very carefully then we can guarantee that $|\mathbf{PR}(\mathcal{A})| = 1$. For a higher-level understanding of what will happen next, the reader may imagine that the ω-chain serves as a "coordinate axis" while various special patterns of cycles play the role of the "coordinates" along this axis. We illustrate the basic idea in the example below.

Example 3.1. For example, suppose in $\mathcal{A}$ we start by producing only 1-cycles (loops):

$$c(s^k(o)) = s^k(o) \text{ for } k = 1, 2, \ldots.$$

Given some other punctual $\mathcal{C}$, we wait for $\mathcal{C}$ to show us two consequent 1-cycles. Either we will discover that $\mathcal{C}$ is not isomorphic to $\mathcal{A}$ or $\mathcal{C}$ will give us two consequent 1-cycles. Once this happens we can switch in $\mathcal{A}$ to a sequence of 1-cycles and 2-cycles:

$$\ldots - 1 - 2 - 1 - 2 - 1 - 2 - 1 - \ldots.$$

We will then use the speed of enumeration of $\mathcal{C}$ and the unary functions p, c, s to locate the exact position of the revealed two consequent cycles of $\mathcal{C}$ relative to its version of the origin o. With some care this idea can be extended to handle all primitive recursive structures $\mathcal{C}_0, \mathcal{C}_1, \ldots$.

Now, based on this intuition, we will give an informal overview of the proof of the theorem. Imagine for a moment that the structure $\mathcal{A}$ constructed above did not have any cycles and was simply the successor structure (ω, s). To produce a copy $\mathcal{T}$ of $\mathcal{A}$ not punctually isomorphic to the standard term-presentation $\mathcal{B}$ of (ω, s), we could do the following naive diagonalization. In $\mathcal{T}$, introduce a new fresh element v but keep it disconnected from the generator o. We do not have to punctually declare k such that $s^k(o) = v$, so we will opt to pick such a k very late. We wait until $p_0(v) \downarrow = s^j(o)$ in $\mathcal{B}$, and then make sure $k > j$. Then we repeat with new witnesses $v', v'', \dots$ for $p_1, p_2, \dots$ to diagonalise against all potential primitive recursive isomorphisms. Note that each p_i has to be total, and also that there is no delay in building $\mathcal{T}$, for we can freely define $s^t(v)$ even though we do not know the term for v itself. The intuition here is that x may look "non-standard" for as long as we wish.

Of course, our structure is not simply (ω, s) (and it cannot be, for the punctual degrees of (ω, s) are upwards dense). Indeed, we are dealing with a chain of cycles. Imagine that we wish to put a fresh v into $\mathcal{T}$ for the sake of diagonalization. Not only we have to start promptly producing elements of the form $s^i(v)$ in $\mathcal{T}$, but also we must punctually decide on the sizes of c-cycles around these elements. Recall that the purpose of these cycles was to press $(\mathcal{C}_e)_{e \le t}$ to reveal themselves in a predictable way, where t depends on the current stage (and is slowly increasing with time). To punctually reconstruct the unique isomorphism from a fixed $\mathcal{C}_e$ onto $\mathcal{T}$ we could try to use the unary function p to go back to the generator o of $\mathcal{C}_e$ and then work from there to find the "coordinates" of the revealed pattern, but this p will no longer help.

Clearly, there is a conflict between the naive diagonalization strategy and the pressing strategy, for we intend to keep parts of the chain disconnected from o. In particular, we cannot possibly hope to use the generator o to reconstruct the "coordinates" of the revealed pattern, because it may well be corresponding to one of the "non-standard" elements generated by the current diagonalization witness v. There will be no uniformly primitive recursive bound on how long it will take for v to be declared a term of o. This time non-uniformity tricks will not help.

Instead, whenever we introduce a fresh diagonalization witness v, the s-chain generated by v will be associated with some fresh array of admissible cycles distinct from those used so far in the chain $s^i(o)$, $i \le t$, seen so far in $\mathcal{B}$. For example, in the term subalgebra $\langle o \rangle$ we will be operating with 1-cycles, 2-cycles, and 3-cycles, while in the term subalgebra $\langle v \rangle$ we will be using exclusively 7-, 9- and 11-cycles. Using p, we will map each such $c^m(s^k(v))$ back to v (not to o as before). Otherwise the strategies restricted to $c^m(s^k(v))$ will be roughly similar to those outlined above for pressing $\mathcal{C}_e$.

Now, when $\mathcal{C}_e$ reveals a part of itself, we will be checking on the sizes of the c-cycles which are revealed. If the sizes correspond to those currently built in the "standard" chain, we will act (roughly) as before. If the sizes look too large, then we may have an opportunity to ensure $\mathcal{C}_e \not\cong \mathcal{A}$ by keeping all our currently admissible c-cycles smaller than the longest c-chain $\mathcal{C}_e$; this is the same as before. Finally, if these cycles have sizes which belong to the chain generated by v, we will find the position of these elements *relative to* v. This will allow us to map these revealed elements of $\mathcal{C}_e$ to the respective elements in $\mathcal{T}$ without knowing the term for v.

Unfortunately, the strategy described above is not easy to formally implement. Apart from the evident priority conflicts, there will also be rather subtle timing issues. For example, what if $\mathcal{C}_e$ responds almost at the same stage as when we decide to declare $s^k(o) = v$? Whenever we declare $s^k(o) = v$ we must also make further alterations in the way we build $\mathcal{T}$, and this will potentially result in a delay in the definition of the isomorphism. The underlying idea is, of course, to produce a *uniform* bound on whatever delays that may occur in $\mathcal{T}$. Furthermore, each isomorphism-building strategy will typically have to deal with a finite non-uniform parameter which will depend on injury and on the actions of the higher priority diagonalization and pressing strategies. All these subtleties will be handled in the formal proof below. Although this is just a Π^0_2-argument and the injury will be merely finite, there will be much combinatorics to sort out.

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

- 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 $f \in \{s_n, c_n, p_n, r_n\}$.

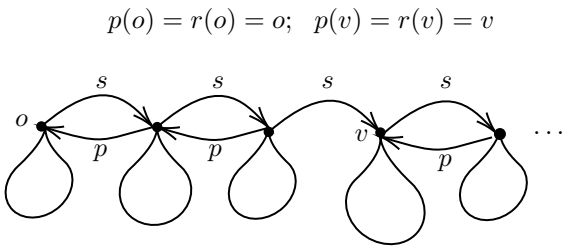

FIGURE 1. The structure $\mathcal{A}$ with two roots o and v shown.

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: $R_0 < P_0 < R_1 < P_1 < \ldots$. 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. These calculations will 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 diagonalise 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*

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

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. We build a (partial) function θ from ω to ω. Our goal is the following: If the structure $\mathcal{C}_e$ is isomorphic to $\mathcal{T}$, then θ will be a primitive recursive isomorphism acting from $\mathcal{C}_e$ onto $\mathcal{T}$.

The basic idea behind the construction of θ is *pattern-switching*. For the sake of simplicity, first we consider a *toy situation*: assume that R_e does not care at all about other strategies R_l, where $l \neq e$. Then *every c-cycle*, which is not an island tag, will have length equal to either $3e+1$ or $3e+2$.

We use the following *patterns*:

- The simplest pattern is denoted by $[3e+1]$. When we say that we iterate (or produce, or propagate) the pattern $[3e+1]$, this means that whenever we add a fresh c-cycle C (which is not an island tag) into $\mathcal{B}$ or $\mathcal{T}$, the cycle C has length $3e+1$.
- A little bit more complicated pattern is denoted by $[3e+1, 3e+2]$. When we write that we iterate the pattern $[3e+1, 3e+2]$, then this means the following: when we attach fresh c-cycles either to the main part B' of $\mathcal{T}$, or to the current island I inside $\mathcal{T}$, we alternate lengths between $3e+1$ and $3e+2$. For example, attaching cycles to B' looks like this:
  - Assume that n is the largest number such that $s_T^n(o)$ is currently defined, and the length of the c_T-cycle of $s_T^n(o)$ is equal to $3e+2$. Then choose $s_T^{n+1}(o)$ as an element of a fresh c-cycle of size $3e+1$.
  - Pick $s_T^{n+2}(o)$ as an element of a fresh cycle of size $3e+2$.
  - Pick $s_T^{n+3}(o)$ as an element of a cycle of size $3e+1$.
  - Pick $s_T^{n+4}(o)$ as an element of a cycle of size $3e+2$; etc.

  Surely, a similar construction proceeds inside the structure $\mathcal{B}$.
- Iterating the pattern $[3e+1, 3e+2, 3e+2]$ means the following: Inside both the main part B' and the island I, we choose the lengths of fresh c-cycles as follows:

$$3e+1, 3e+2, 3e+2, 3e+1, 3e+2, 3e+2, 3e+1, 3e+2, 3e+2, \ldots.$$

  Again, similar actions are taken for $\mathcal{B}$.
- In a similar way, one can define what does it mean to produce the pattern

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

  where k is an arbitrary natural number.

When we talk about a *pattern-switching*, we always mean that we switch from a pattern

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

which is currently propagated, to the pattern

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

The switching can be done pretty straightforwardly: For example, suppose that we want to switch from the pattern $[3e+1, 3e+2]$ to $[3e+1, 3e+2, 3e+2]$. Let n be the largest number such that the value $v := s_T^n(o)$ is currently defined, and assume that the c-cycle of v has length $3e+1$. Then we just choose the lengths of fresh c-cycles like this:

$$3e+2, 3e+2, 3e+1, 3e+2, 3e+2, 3e+1, 3e+2, 3e+2, \ldots.$$

The pattern-switching will help us to ensure that the strategy R_e will always have one of the following two outcomes:

(i) If the pattern-switching happens only finitely many times, then the structure $\mathcal{C}_e$ is not isomorphic to $\mathcal{T}$.
(ii) If the pattern-switching happens infinitely many times, then the constructed function θ is a punctual isomorphism from $\mathcal{C}_e$ onto $\mathcal{T}$.

We build θ step-by-step: the values $\theta(0)$, $\theta(1)$, $\theta(2)$, ..., $\theta(n)$, $\theta(n+1)$, ... are computed, in turns. While the value $\theta(0)$ is still being computed, the construction propagates the simplest pattern $[3e+1]$.

The value $\theta(0)$ is defined as follows:

(1) Compute the root $r_e(0)$ and find its island tag l^0, i.e. the length of the c_e-cycle of $r_e(0)$. It can be the case that inside $\mathcal{C}_e$, the root $r_e(0)$ is a part of an infinite c_e-chain and hence, the value l^0 will be never computed. Clearly, in this case $\mathcal{C}_e$ cannot be a copy of $\mathcal{T}$.
(2) While we are waiting for the value l^0 to be computed, each of the lower priority strategies P_i, $i \geq e$, follows the restrictions set forth by R_e: The strategy P_i cannot become active until one of the following conditions is satisfied.
  (2.1) The tag l^0 is defined and $l^0 \in L(P_i)$, i.e. l^0 is an acceptable tag for P_i.
  (2.2) l^0 is defined and l^0 is strictly less that every element from $L(P_i)$.
  (2.3) The current value of l^0 is greater than every element from $L(P_i)$. Here by the value of l^0 at a stage s we mean the maximal number $m \leq s$ such that every value $c_e^i(r_e(0))$, $i \leq m$, can be computed in s steps of computation. Obviously, here we assume that all $c_e^i(r_e(0))$, $i \leq m$, are pairwise different.

  In Case (2.1), we will never use the value l^0 as an island tag for P_i, and this will ensure that $\mathcal{C}_e \ncong \mathcal{T}$. In each of Cases (2.2) and (2.3), the value l^0 is not related in any way to the acceptable island tags of P_i.
(3) After l^0 is computed, there are three possible cases:
  (A) l^0 does not belong to any $L(P_i)$, where $-1 \leq i < \omega$. Recall that we assume that $L(P_{-1}) = \{3\}$. In this case, it is clear that $\mathcal{C}_e$ is not isomorphic to $\mathcal{T}$. Hence, one can just stop defining the function θ. The pattern $[3e+1]$ is propagated forever.
  (B) l^0 belongs to some $L(P_i)$, where P_i is of lower priority, i.e. $i \geq e$. Then we forbid P_i to use l^0 as its island tag. Hence, by the construction, $\mathcal{T}$ does not possess a c-cycle of size l^0, but $\mathcal{C}_e$ contains such cycle. Therefore, $\mathcal{C}_e \ncong \mathcal{T}$. Again, here we stop defining θ, and we iterate $[3e+1]$ forever.
  (C) l^0 is an acceptable tag for some P_i of higher priority, i.e. $i < e$. For now, just continue iterating $[3e+1]$.
(4) From now on, assume that the case (C) above holds. Wait until (the atomic diagram of) the structure $\mathcal{C}_e$ shows computations witnessing one of the following outcomes:
  - 0 belongs to the c_e-cycle of the root $r_e(0)$;
  - 0 belongs to the c_e-cycle of the element $s_e(r_e(0))$;
  - both c_e-cycles of 0 and $p_e(0)$ have length $3e+1$.

  Essentially, we want to see that either 0 is situated close enough to its root $r_e(0)$, or there are two adjacent cycles of size $3e+1$ in the vicinity of 0.

Recall that our construction is still propagating $[3e+1]$, so if the structure $\mathcal{C}_e$ "wants" to be a copy of $\mathcal{T}$, then it *should* provide us with the desired computations. Let u_0 be the number of steps needed to provide these computations.

When the u_0-th step of the construction finishes, we immediately switch from the pattern $[3e+1]$ to $[3e+1, 3e+2]$. This action ensures the following fact: If u_0 is given, then one can promptly find the value of $\theta(0)$.

Indeed, the construction will guarantee that by the end of step u_0, the structure $\mathcal{T}[u_0]$ will already contain a root r^0 with the island tag l^0. Hence, in order to find the desired image $\theta(0)$, it is sufficient to consider the following finite structures:

- Inside $\mathcal{C}_e$, we choose elements $r_e(0), s_e(r_e(0)), s_e^2(r_e(0)), \dots, s_e^{u_0}(r_e(0))$ and the c_e-cycles generated by these elements. Denote this structure by $C[0]$.
- Inside $\mathcal{T}$, we find elements $r^0, s(r^0), s^2(r^0), \dots, s^{u_0}(r^0)$ and the c-cycles of these elements. Denote the corresponding structure by $T[0]$.

Recall that after the step u_0, we never add adjacent $(3e+1)$-cycles into $\mathcal{T}$. Hence, the number 0 must belong to $C[0]$. On the other hand, the island tag of r^0 is equal to l^0. Hence, if $\mathcal{C}_e \cong \mathcal{T}$, then the unique isomorphism acting from the (rigid) $\mathcal{C}_e$ onto $\mathcal{T}$ must map $C[0]$ onto $T[0]$. Thus, the desired isomorphic image $\theta(0)$ can be recovered by promptly searching inside the structure $T[0]$.

When the value $\theta(0)$ is defined, we start computing $\theta(1)$. First, we compute the root $r_e(1)$ and we find its island tag l^1. While waiting for l^1 to be computed, R_e restricts each of the strategies P_i, $i \geq e+1$. Note that here we *do not* put restrictions on the strategy P_e. A strategy P_i, where $i \geq e+1$, cannot become active until one of the following conditions happens to be satisfied:

(2.1) l^1 is defined and $l^1 \in L(P_i)$;
(2.2) l^1 is defined and $l^1 < \min(L(P_i))$;
(2.3) the current value of l^1 is greater than $\max(L(P_i))$.

While l^1 is being computed, the construction continues iterating the pattern $[3e+1, 3e+2]$.

After l^1 is computed, we have four alternatives:

(A) l^1 does not belong to any $L(P_i)$. Then $\mathcal{C}_e \not\cong \mathcal{T}$, and we iterate the same pattern forever.
(B) l^1 belongs to some $L(P_i)$, where $i \geq e+1$. Then we forbid l^1 to be used as an island tag, and this will guarantee that $\mathcal{C}_e \not\cong \mathcal{T}$.
(C) The case when $l^1 \in L(P_e)$ is more delicate. Here we have two variants:
  (C.1) If the strategy P_e is still in one of states S1 or S2, i.e. P_e is unsure about its island tag, then we just forbid l^1 to be used by P_e. This action is well-behaved, since R_e forbids P_e to use *only one* tag l^1, recall that l^0 has nothing to do with P_e. As in (B), this will ensure that $\mathcal{C}_e \not\cong \mathcal{T}$.
  (C.2) If P_e is already in one of states S3 or S4, then we do not want to forbid l^1. Here we emphasize the following: By the moment t^1 when the computations of l^1 had been finished, our structure $\mathcal{T}[t_1]$ *already contained* island tag l^1.
(D) l^1 is an element of $L(P_i)$, where $i < e$.

We assume that one of the cases (C.2) or (D) holds. We wait until our structure $\mathcal{C}_e$ shows computations witnessing one of the following two outcomes:

- 1 belongs to the c_e-cycle of one of the elements $r_e(1)$, $s_e(r_e(1))$, $s_e^2(r_e(1))$, $s_e^3(r_e(1))$;
- among the c_e-cycles of the elements $p_e^3(1)$, $p_e^2(1)$, $p_e(1)$, 1, we can find either two adjacent $(3e+1)$-cycles, or three adjacent cycles of lengths $3e+1$, $3e+2$, and $3e+1$, respectively.

Recall that the construction is propagating the pattern $[3e+1, 3e+2]$. If $\mathcal{C}_e \cong \mathcal{T}$, then the desired computations will be eventually provided. Let u_1 be the number of steps needed to finish these computations.

When the u_1-th step of the construction finishes, we switch from iterating $[3e+1, 3e+2]$ to $[3e+1, 3e+2, 3e+2]$. We claim that given the value u_1, one can punctually compute $\theta(1)$. Indeed, this follows from the following two facts:

(1) By the end of step u_1, we will already have a root r^1, which has the island tag l^1 inside $\mathcal{T}[u_1]$.
(2) After step u_1, between any two fresh c_T-cycles of size $3e+1$, there will be at least two c_T-cycles of size $3e+2$.

Now assume that the values $\theta(0), \theta(1), \ldots, \theta(x-1)$ have been already calculated. When we start computing $\theta(x)$, our construction propagates the pattern

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

We find the root $r_e(x)$ and its island tag l^x. While waiting for l^x to be defined, R_e restricts each of the strategies P_i, $i \geq e+x$: any such P_i cannot become active until one of the following conditions is satisfied.

(2.1) l^x is defined and $l^x \in L(P_i)$;
(2.2) l^x is defined and $l^x < \min(L(P_i))$;
(2.3) the current value of l^x is greater than $\max(L(P_i))$.

While l^x is being computed, the construction iterates the pattern (3.1).

After l^x is computed, again, we have four alternatives:

(A) l^x does not belong to any $L(P_i)$. Then $\mathcal{C}_e \ncong \mathcal{T}$.
(B) l^x belongs to some $L(P_i)$, where $i \geq e+x$. Then we forbid l^x to be used as an island tag.
(C) $l^x \in L(P_i)$, where $e \leq i < e+x$. Then one of the following two cases holds:
  (C.1) If P_i is in one of states S1 or S2, then we forbid l^x to be used by P_i.
  (C.2) If P_i is in one of states S3 or S4, then we do not forbid l^x.
(D) l^x is an element of $L(P_i)$, where $i < e$.

Assume that one of the cases (C.2) or (D) holds. We wait until our structure $\mathcal{C}_e$ shows computations witnessing one of the following two results:

- x belongs to the c_e-cycle of one of the elements $r_e(x)$, $s_e(r_e(x))$, $s_e^2(r_e(x))$, $\ldots$, $s_e^{2x+1}(r_e(x))$;
- among the c_e-cycles of the elements $p_e^{2x+1}(x)$, $p_e^{2x}(x)$, $\ldots$, $p_e^2(x)$, $p_e(x)$, x, one can find adjacent cycles such that their lengths induce one of the

following tuples:

$$(3e+1, 3e+1);\ (3e+1, 3e+2, 3e+1);\ (3e+1, 3e+2, 3e+2, 3e+1);\ \ldots;$$
$$(3e+1, \underbrace{3e+2, \ldots, 3e+2}_{x \text{ times}}, 3e+1).$$

If $\mathcal{C}_e \cong \mathcal{T}$, then the required computations eventually will appear. By u_x we denote the number of steps needed to finish the computations. When the u_x-th step of the construction finishes, we switch from iterating (3.1) to

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

Again, one can argue that given u_x, one can promptly compute the value $\theta(x)$. Hence, in order to establish the existence of the punctual isomorphism θ, it will be sufficient to show that the sequence $(u_x)_{x\in\omega}$ is primitive recursive in x. The reason behind this primitive recursiveness will be elaborated in the construction and verification.

Recall that the described setting is merely a toy situation. Indeed, things becomes more involved, when different strategies R_e, $e \in \omega$, mingle with each other. Nevertheless, this is not a very hard obstacle: different strategies R_e will use *different coding locations*.

We give a rough outline of these coding locations: For the sake of simplicity, assume that we build only the main part B' of the structure $\mathcal{T}$, and we do not glue any islands to B'. The root of B' will be denoted by r.

Clearly, in this case, all information about a given element $x \in \mathcal{T}$ is completely defined by its island coordinates: recall that this is a pair (m, k), where the element $s^m(o)$ belongs to the c_T-cycle of x, and k is the least with $c^k(s^m(o)) = x$.

We use all cycles with coordinates $(2k, \cdot)$, $k \in \omega$, as coding places for the strategy R_0. The strategy R_1 will use the cycles with coordinates $(4k+1, \cdot)$, $k \in \omega$. The strategy R_2 uses $(8k+3, \cdot)$. In general, a strategy R_e employs the coordinates $(\langle e, k\rangle, \cdot)$, where $k \in \omega$.

What does it mean in practice? The actions of our strategy R_e should be modified according to its choice of coding places. First, the patterns are propagated only in appropriate places: For example, iteration of a pattern $[3e+1, 3e+2]$ means that one proceeds as follows:

- the element $s^{\langle e,0\rangle}(r)$ will be a part of a c-cycle of size $(3e+1)$;
- the c-cycle of $s^{\langle e,1\rangle}(r)$ will have size $(3e+2)$;
- the c-cycle of $s^{\langle e,2\rangle}(r)$ will be of size $(3e+1)$;
- the c-cycle of $s^{\langle e,3\rangle}(r)$ will have size $(3e+2)$;
- $\ldots$

This kind of iteration helps R_e to avoid messing with the c-cycles belonging to other strategies R_j, $j \neq e$.

Second, the construction of $\theta(x)$ should be also changed: The c-cycles of size $\ell \in \{3e+1, 3e+2\}$ are much more disperse now, so we have to make a conscious effort even just to *find these cycles*.

Recall that when we computed $\theta(0)$, we wanted to find two *adjacent* $(3e+1)$-cycles. Obviously, this search must be modified. The modification is essentially as follows. If we consider the values 0, $p_e(0)$, $p_e^2(0)$, $\ldots$, $p_e^{2^{e+1}}(0)$, then:

- either one of these elements is already the root of the main part of $\mathcal{C}_e$, or

- at least one of these elements must have a c_e-cycle of size $3e+1$ (otherwise, the structure $\mathcal{C}_e$ is not a copy of $\mathcal{T}$, and we can stop caring about building θ).

If one of the elements is the root, then one can already find $\theta(0)$ (in a prompt way). Otherwise, let y_0 be the (first found) element such that

$$y_0 \in \{0, p_e(0), p_e^2(0), \dots, p_e^{2^{e+1}}(0)\},$$

and the c_e-cycle of y_0 has size $3e+1$. We find the value $y_1 := p_e^{2^{e+1}}(y_0)$. Again,

- either y_1 is already the root, or
- y_1 has a c_e-cycle of size $3e+1$. Indeed, recall that since the start, the construction has been propagating the pattern $[3e+1]$. Thus, if the size is different from $3e+1$, then $\mathcal{C}_e$ is not a copy of $\mathcal{T}$.

Now the value u_0 is defined as the number of steps needed to finish the described search procedure *successfully*: If $\mathcal{C}_e \cong \mathcal{T}$, then we always will have a successful finish — by this we mean that either at some point, we will prematurely reach the root of $\mathcal{C}_e$, or we will witness two desired $(3e+1)$-cycles.

When u_0 is successfully computed, we immediately switch to the pattern $[3e+1, 3e+2]$ (again, putting it in appropriate places). An argument, similar to the one dealing with a single R_e, shows that given u_0, one can promptly find $\theta(0)$.

In general, the procedure of finding $\theta(x)$ looks like this:

(a) Find some $y_0 \in \{x, p_e(x), p_e^2(x), \dots, p_e^{(2x+1)\cdot 2^{e+1}}(x)\}$ such that its c_e-cycle has size $3e+1$.

(b) Find the values $y_{i+1} := p_e^{2^{e+1}}(y_i)$, for $i \le x$. If among the lengths of the c_e-cycles of $y_{x+1}, y_x, \dots, y_1, y_0$, one can find a consecutive tuple having one of the forms

$$(3e+1, 3e+1);\ (3e+1, 3e+2, 3e+1);\ (3e+1, 3e+2, 3e+2, 3e+1);\ \dots;$$
$$(3e+1, \underbrace{3e+2, \dots, 3e+2}_{x \text{ times}}, 3e+1),$$

then declare that the u_x-procedure is finished succesfully.

Note that while processing one of the computations above, one can prematurely reach the root, i.e. discover that, say, $p_e^3(y_2) = r(x)$. In this case, we also say that the u_x-procedure finished succesfully.

Recall that before the value $\theta(x)$ is computed, the iterated pattern is

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

Hence, if $\mathcal{C}_e \cong \mathcal{T}$, then the u_x-procedure must finish successfully, and one can define u_x as the number of steps needed to finish the procedure. When the u_x-procedure finishes successfully, we immediately switch to the pattern

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

Again, given u_x, one can quickly compute the value $\theta(x)$.

This finishes the description of a strategy R_e in isolation. The details considering the prompt computations of u_x, $x \in \omega$, will be given in the formal construction and the verification.

(In the construction below, we view every 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.)

Construction.

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.

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

We want our strategy P_i to move on, and to enter state S3 (i.e. to choose its own island tag). In order to proceed in a safe way, we have to respect all restrictions which are set forth by the higher priority strategies R_j, $j \leq i$. Recall that the informal description of R_e given above tells us the following:

- A strategy R_j, $j \leq i$, can forbid P_i to use some tag $l \in L(P_i)$ only while R_j is computing the values $\theta_j(0)$, $\theta_j(1)$, …, $\theta_j(i-j)$. Here θ_j is a (potential) isomorphism from $\mathcal{C}_j$ onto $\mathcal{T}$. After the value $\theta_j(i-j)$ is computed, R_j does not impose any restrictions on P_i.
- Assume that by a stage t, the strategy R_j has already computed, say, $\theta_j(0)$ and $\theta_j(1)$, and the corresponding island tags (denoted by l^0 and l^1 in the description of R_e) have nothing to do with P_i: more formally, this means $l^0, l^1 \notin L(P_i)$. If by the stage t, we have already witnessed that the island tag l^2 inside $\mathcal{C}_j$ is *very large*, then it is safe to assume that R_j will not put any future restrictions on us, and hence, P_i can choose its island tag.

  Here the tag l^2 is *very large* if the following holds: After t steps of computation, all the values

  $$r_j(2), c_j(r_j(2)), c_j^2(r_j(2)), \dots, c_j^M(r_j(2))$$

  have been computed, and they are pairwise different. Informally, this means that the island tag of $r_e(2)$ is already too big to be acceptable for the strategy P_i.

  Note that here we discussed a simplified version of the argument: In reality, P_i needs to obtain some information about *each* higher priority R_j (not only one R_j), before P_i will be able to safely choose its island tag.

So, in order to help P_i to move on, we make $(s+1)$ steps of the following computations.

($\star$) For each $j \leq i$ and each $x \leq i - j$, compute the values

$$r_j(x), c_j(r_j(x)), c_j^2(r_j(x)), \dots, c_j^M(r_j(x)).$$

After the values have been computed, find a finite set F of *forbidden tags* as follows:

$$v_j := \begin{cases} \text{the least } l \in L(P_i) \text{ such that for some } x \leq i-j & \\ \text{ the } c_j\text{-cycle of } r_j(x) \text{ has length } l, & \text{if such } l \text{ exists;} \\ -1, & \text{otherwise.} \end{cases}$$

$$F := \{v_j : j \leq i,\ v_j \neq -1\}.$$

Note that clearly, the set F (when computed) contains at most $i+1$ elements.

If the computation $(\star)$ does not finish in $(s+1)$ steps, then P_i stays in state S2. Otherwise, it *seems* that P_i is ready to choose its island tag, *but* this choice still requires a little bit of polishing. This polishing will prove to be useful in the verification (see Lemma 3.4). We illustrate the necessity of polishing by the following example.

Example 3.2. Consider the strategy P_1. In order to find the set F for P_1, we need to wait for the following values to be computed:

(i) $r_0(0), c_0(r_0(0)), \ldots, c_0^M(r_0(0))$;
(ii) $r_0(1), c_0(r_0(1)), \ldots, c_0^M(r_0(1))$;
(iii) $r_1(0), c_1(r_1(0)), \ldots, c_1^M(r_1(0))$.

It can be the case that both computations (i) and (ii) are pretty quick: say, in 100 steps they show that inside $\mathcal{C}_0$, the island tags for both 0 and 1 are too big for $L(P_1)$; in other words, 100 steps of computation already witness that $v_0 = -1$. On the other hand, the computations (iii) could be very slow: say, only in 10^{10} steps, we will see that inside $\mathcal{C}_1$, the number $r_1(0)$ is a part of a c-cycle of size M (i.e., witnessing the equality $v_1 = M$ requires a lot of time).

Hence, in 10^{10} steps, we are happy to declare that $F = \{M\}$. Furthermore, we can already conclude that $\mathcal{C}_1$ cannot be a copy of $\mathcal{T}$, since $\mathcal{C}_1$ contains a cycle of size M, but P_1 will never use this size (recall that P_1 is the *only* strategy which can use the tag M). Summarising, we successfully diagonalised against $\mathcal{C}_1$, but we did not obtain any conclusive evidence against $\mathcal{C}_0$.

Nevertheless, it is *possible* that this evidence against $\mathcal{C}_0$ was really close to us: Say, in just 10^3 steps, one could have witnessed that the c_0-cycle of $r_0(2)$ had size $\ell \neq M$ such that $\ell \in L(P_1)$. Therefore, using pretty much *the same* 10^{10} steps, we could have forbidden P_1 to use ℓ, and successfully diagonalise against $\mathcal{C}_0$.

So, in order to avoid situations similar to that of Example 3.2, we proceed as follows. If the computation $(\star)$ finishes in $(s+1)$ steps, then we make an additional *tag-prohibition* routine: Consider each $j \leq i$ with $v_j = -1$. For each $y \leq s$, make $(s+1)$ steps of computing

$$r_j(y), c_j(r_j(y)), c_j^2(r_j(y)), \ldots, c_j^M(r_j(y)).$$

Find the least y, for which these bounded computations witness that $r_j(y)$ has c_j-cycle of a size $\ell_y \in L(P_i)$. If such y exists, then put ℓ_y into the set F of forbidden tags. It is clear that the cardinality of (the modified) F is still at most $i+1$.

At last, after the routine, P_i is allowed to choose its tag. Find the least tag $l \in L(P_i) \setminus F$. We declare l the *island tag* of P_i, and say that P_i is now in state S3 (i.e. active). We initialise 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*.

(0.3) 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 elaborated in Phase 2, this is essentially the pattern-switching introduced in the description of the strategy R_e).

After that proceed to Phase 2.

Phase 2: Growing the structure $\mathcal{B}$.

First, we explain how we define our working patterns. The patterns will help us to prove that the sequence $(u_x)_{x\in\omega}$ from the description of R_e can be computed in a prompt way.

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 $i \leq (2x+1)\cdot 2^{e+1}$ 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 $(2x+1)\cdot 2^{e+1}$ 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 a fixed 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

$$[3e + 1, \underbrace{3e + 2, \ldots, 3e + 2}_{k \text{ times}}] \tag{3.2}$$

  forever. If every $u_{q,l,e}(k)$ stops successfully, then the pattern (3.2) 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. In both $\mathcal{B}$ and $\mathcal{T}$ there is no delay unbounded in the definition of the finitely many unary functions in the language of $\mathcal{A}$. Therefore, both structures $\mathcal{B}$ and $\mathcal{T}$ are punctual.

Before giving the formal lemmas, we emphasise an important feature of the described construction: Without loss of generality, one may assume that a strategy R_e starts working *only after* all strategies P_i, $i < e$, moved to state S3. In other words, we start considering the structure $\mathcal{C}_e$ (and building an isomorphism θ_e from $\mathcal{C}_e$ onto $\mathcal{T}$) only after each P_i, $i < e$, chose its island tag. This assumption essentially slows down the "data stream" describing $\mathcal{C}_e$, but this process does not injure the primitive recursiveness of the function α: roughly speaking, one introduces sufficiently many initial idle steps t, at which the object $\mathcal{C}_e[t]$ is treated as an empty structure; and these steps t propagate the simplest pattern $[3e+1]$ inside both $\mathcal{B}$ and $\mathcal{T}$.

Lemma 3.3. *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 $(\star)$ 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 3.3) and by copying $\mathcal{T}$ from $\mathcal{B}$ at Phase 2.

Lemma 3.4. *Every requirement R_e is satisfied.*

Proof. Suppose that $\mathcal{C}_e$ is isomorphic to $\mathcal{T}$ and θ 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 $e \leq i$ and $x \leq i - e$. Recall that the strategy P_i becomes active only after the computations $(\star)$ are finished. Since $l^x \in L(P_i)$, there exists the least length $l \in L(P_i)$ such that for some $x' \leq i - e$, the c_e-cycle of $r_e(x')$ has length l. This length l belongs to the set of forbidden tags F for the strategy P_i. Thus, $\mathcal{T}$ does not have cycles of size l, but $\mathcal{C}_e$ contains a cycle of size l. Hence, $\mathcal{C}_e \not\cong \mathcal{T}$, and we obtained a contradiction.

Therefore, we have $i < e + x$. 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 $\theta(q)$ of a root q from $\mathcal{C}_e$. First, find the corresponding island tag l^q and the index i_q such that $l^q \in L(P_{i_q})$. Suppose that t^* is the number of steps needed to see that q belongs to a c_e-cycle of size l^q inside $\mathcal{C}_e$. Without loss of generality, one may assume that $t^* > e + q > i_q$. We argue that the element $\theta(q)$ must already belong to $\mathcal{T}[t^*]$.

Towards contradiction, assume that $\theta(q) \notin \mathcal{T}[t^*]$. Consider Phase 0.2 of the stage t^*. Since $\theta(q) \notin \mathcal{T}[t^*]$, 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.

If $e > i_q$, then without loss of generality, one may assume that t^* is strictly greater than the number of steps needed to compute, using the procedure $(\star)$, the set of forbidden tags F for the strategy P_{i_q}. Indeed, this assumption can be achieved by "slowing down" discussed at the beginning of verification: one starts considering the structure $\mathcal{C}_e$ only after each strategy P_i, $i < e$, already chose its island tag. Hence, in this case, $\theta(q)$ already belongs to $\mathcal{T}[t^*]$.

Thus, here we need to consider only the case when $e \leq i_q$. Recall that at the beginning of Phase 0.2 of t^*, the strategy P_{i_q} was in state S2. This implies that the computations $(\star)$ for P_{i_q} require at least $(t^* + 1)$ steps to finish. Let denote this particular number of steps by $(t' + 1)$. At the stage t' of the construction, the strategy P_{i_q} finishes its computations $(\star)$. Before P_{i_q} defines its island tag, it makes additional tag-prohibition routine: Since $t' \geq t^* > q$, in t' steps of computation, one can witness that the element $r_e(q)$ already has a c_e-cycle of size l^q. This ensures that R_e forbids P_{i_q} to use some island tag $l' \in L(P_{i_q})$, and hence, $\mathcal{C}_e$ cannot be isomorphic to $\mathcal{T}$ — a contradiction.

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

Finally, for an arbitrary natural number x, we show how to compute its image $\theta(x)$ 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 $\theta(x) := c_T^k(\theta(q))$.

Suppose that $p_e(s_e(x)) \neq q$. Let (m, k) be the island coordinates of the element x. In order to compute $\theta(x)$, 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 deduce that $k \leq 3i + 2$, and after finding $\theta(q)$, one can determine the image $\theta(x)$ 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 contradicts the condition $\mathcal{C}_e \cong \mathcal{T}$. Therefore, $u_{q,l,e}(z)$ must stop successfully for all z.

We claim that the first island coordinate m of the element x is at most

$$u_{q,l,e}(x) + (3x+2)\cdot 2^{e+1}.$$

Here the additional summand $(3x+2)\cdot 2^{e+1}$ appears because of the following (a little bit informal) considerations:

- In the procedure of calculating $u_{q,l,e}(x)$, one has to apply p_e at most $(2x+1)\cdot 2^{e+1}$ times before finding y_0.
- Recall that $y_1 = p_e^{2^{e+1}}(y_0)$, $y_2 = p_e^{2^{e+1}}(y_1)$, $\ldots$, $y_{x+1} = p_e^{2^{e+1}}(y_x)$. Each of these equalities provides precisely 2^{e+1} applications of the function p_e.

Towards contradiction, assume that $m > u_{q,l,e}(x)+(3x+2)\cdot 2^{e+1}$. 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 satisfies the following conditions:

- $z \le x$;
- each of y_0 and y_{z+1} belong to a c_e-cycle of size $3e+1$;
- each y_u, $1 \le u \le z$, has a c_e-cycle of size $3e+2$; and
- the number $\langle e, j\rangle$, i.e. the first island coordinate of y_{z+1}, is strictly greater than $u_{q,l,e}(x)$.

This contradicts the following: after $u_{q,l,e}(x)$, our construction propagates only patterns

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

where $v \ge x+1$. Hence, $m \le u_{q,l,e}(x) + (3x+2)\cdot 2^{e+1}$, and the island coordinates of x can be computed in a primitive recursive way. Lemma 3.4 is proved. □

Theorem 1.4 is proved. □

4. Proof of Theorem 1.7

Proof idea. To understand and appreciate this proof idea the reader must first familiarise themselves with the proof of the previous theorem. This time, the structure will consist of two ω-chains, one generated by a function 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 "island tags". It will be playing the role of the coordinate axis in the structure, with patterns of cycles playing the role of the coordinates. It will be also used to press only the honestly 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.

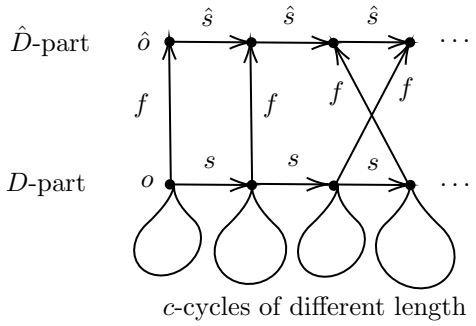

FIGURE 2. The structure $\mathcal{A}$ from Theorem 1.7

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 an 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 promptly 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). 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
(2) we set

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

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 $|\mathbf{PR}(\mathcal{A})| > 1$.
(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 $\theta = F^{-1}$ 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 heavily exploit this fact, and we *delay connecting* 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 the elements of 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 $\widehat{D}(\mathcal{B}) \cup \{o_B\}$, and given $y \in \widehat{D}(\mathcal{B})$, we can quickly find the unique x such that $f_B(x) = y$.

The item (b) above can be obtained via 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 $f_B(x) := y$, and all for other elements x' of the cycle we set $f_B(x') = o$. After that, we extend the non-island part of $\mathcal{T}$ to match the current $\mathcal{B}$, with the corresponding extension to the definition of f_T. The described action ensures 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, the elements $\widehat{s}^k(w_e)$ and $\widehat{s}^{-k}(w_e)$ are the latest values that have been 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 $range(f_T)$.

We emphasize the following feature of the construction: The elements

$$s^2(o), s^4(o), s^6(o), \ldots, s^{2k}(o), \ldots$$

will be used to satisfy the strategies P_i, $i \in \omega$, — for each of these elements, its c_T-cycle will have an *even* length. Odd lengths of c-cycles will be used to satisfy Q_j, $j \in \omega$.

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. Without loss of generality, 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 copies of the P_e-island and 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 the preimage $f_B^{-1}(h_e(w_e))$ lies in a c_B-cycle of length $l' \neq l_e$ (recall that the length l_e, which we used inside $\mathcal{T}$, was chosen fresh). 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.

Again, the idea behind building a (potential) isomorphism $F\colon \mathcal{C}_e \to_{\cong} \mathcal{B}_g$ is *pattern-switching*. We use the patterns

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

where $k \in \omega$ (recall that c-cycles of even length are already reserved for satisfying the strategies P_i). Before the value $F(0)$ is computed, we propagate the simplest pattern $[4e+1]$.

Consider the element 0 from $\mathcal{C}_e$. Calculate the value $\widehat{r}_e(0)$. If $\widehat{r}_e(0) \notin \{o_e, \widehat{o}_e\}$, then trivially $\mathcal{C}_e$ cannot be a copy of $\mathcal{B}_g$. Therefore, without loss of generality, we may assume that $\widehat{r}_e(0) \in \{o_e, \widehat{o}_e\}$, and one can quickly determine whether 0 belongs to $D(\mathcal{C}_e)$ or to $\widehat{D}(\mathcal{C}_e)$.

We want to quickly compute the value $F(0)$. Note that one can promptly find a new value $x^{\#}$, which is defined as follows:

(1) If $0 \in D(\mathcal{C}_e)$, then set $x^{\#} := 0$.
(2) If $0 \in \widehat{D}(\mathcal{C}_e)$, then define $x^{\#} := g_e(0)$. In this case, if $\mathcal{C}_e \cong \mathcal{B}_g$, then 0 should belong to $range(f_e)$ (see the discussion at the end of the overview), and $f_e(x^{\#})$ must be equal to 0.

The element $x^{\#}$ always belongs to $D(\mathcal{C}_e)$ (if it is not the case, 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 $F(0)$. Therefore, for the sake of simplicity, we will assume that $x^{\#} = 0 \in D(\mathcal{C}_e)$.

Apply s_e to 0 at most 2^{e+2} times to find an element y_0 belonging to a c_e-cycle of size $4e+1$. If one compares this action with the corresponding action of the strategy R_e in Theorem 1.4, one can immediately see some subtle differences:

- The strategy R_e applied the *predecessor* function p_e, while here the successor function s_e is used. Informally speaking, the reason behind this is as follows: Here we do not use the island tags "glued" to the roots, so we do not care much about the distance from x to its root. Hence, in order to "catch" the (currently propagating) pattern, it is sufficient just to go forward, along s_e.
- R_e applied p_e at most 2^{e+1} times, but here we use the number 2^{e+2} instead. The reason behind this is pretty simple: recall that all *even* coordinates are already reserved by P_i, $i \in \omega$, for their own needs.

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 the described procedure takes u_0 steps to finish. The procedure finishes *successfully* if we found both numbers y_0 and y_1, and each of them is a part of a c_e-cycle of size $4e+1$. Otherwise, the procedure finishes *unsuccessfully*. In case of unsuccessful finish, it is clear that $\mathcal{C}_e \not\cong \mathcal{B}_g$, and one just continues iterating the pattern $[4e+1]$ forever.

While we are waiting for the number u_0 to be calculated, (as per usual) 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 (unique) number such that the element $s_B^n(o)$ is a part of C. The length l of this C is defined as follows:
  - If n is even (or in other words, $n = \langle 0, m\rangle$ for some m), then $l := 2$.
  - If $n = \langle e+1, m\rangle$ for some m, then $l := 4e+1$.

  Informally speaking, we iterate the pattern $[4e+1]$ in appropriate coding places. In general, if $n = \langle j+1, m\rangle$, then the corresponding length l will be dictated by the current behavior of the strategy Q_j.
- 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 the current $\mathcal{B}$.

If the u_0-procedure finishes successfully, say, at stage t, then we immediately switch to iterating the pattern $[4e+1, 4e+3]$. This ensures that $\mathcal{B}$ will not get fresh adjacent c-cycles of size $4e+1$. Thus, given the value u_0, one can quickly compute $F(0)$: Indeed, the argument similar to that of the strategy R_e (in Theorem 1.4) shows that $F(0)$ can be found by promptly searching inside the finite structure $\mathcal{B}[u_0]$.

After $F(0)$ has been computed, we want to find $F(1)$. Again, for simplicity, we deal with the case $1 \in D(\mathcal{C}_e)$. Apply s_e to 1 at most $3 \cdot 2^{e+2}$ times to find an element y_0 belonging to a c_e-cycle of size $4e+1$. When y_0 is found, compute $y_1 = s_e^{2^{e+2}}(y_0)$ 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 u_1-procedure to finish, grow $\mathcal{B}$ and $\mathcal{T}$ as above, modulo the following modification: We alternate between c_B-cycles of size $4e+1$ and $4e+3$ for elements $s_B^{\langle e+1,m\rangle}(o)$:

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

In other words, the pattern $[4e+1, 4e+3]$ is being propagated.

Again, if the u_1-procedure finishes unsuccessully, then we continue putting the pattern $[4e+1, 4e+3]$, and $\mathcal{C}_e$ is not a copy of our structure. If u_1 is finished successfully, then we switch to the pattern $[4e+1, 4e+3, 4e+3]$. The value $F(1)$ can be promptly recovered via considering $\mathcal{B}[u_1]$. After $F(1)$ is found, we start computing $F(2)$.

We describe the general procedure of computing $F(x)$. Consider $x > 1$. Apply s_e to x at most $(2x+1) \cdot 2^{e+2}$ 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 $0 \le i \le x$. Check whether one of the elements $y_1, y_2, \ldots, y_{x+1}$ belongs to a c_e-cycle of size $4e+1$. If it is the case, then we say that the u_x-procedure finishes *successfully*.

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

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

in appropriate coding places. If the u_x-procedure finishes unsuccessfully, then we continue iterating this pattern forever. If u_x is computed successfully, then we switch to the next pattern

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

Given u_x, one can recover $F(x)$ in a punctual way.

Summarizing, the calculations of u_x, $x \in \omega$, will allow us to quickly compute the isomorphism $F \colon \mathcal{C}_e \to_{\cong} \mathcal{B}_g$.

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 the verification of Theorem 1.4. □

References

[AK00] C. Ash and J. Knight. *Computable structures and the hyperarithmetical hierarchy*, volume 144 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Co., Amsterdam, 2000.

[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.

[Ala17] P. E. Alaev. Structures computable in polynomial time. I. *Algebra Logic*, 55(6):421–435, 2017.

[Ala18a] P. E. Alaev. Categoricity for primitive recursive and polynomial Boolean algebras. *Algebra Logic*, 57(4):251–274, 2018.

[Ala18b] P. E. Alaev. Structures computable in polynomial time. II. *Algebra Logic*, 56(6):429–442, 2018.

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

[BDKM19] N. Bazhenov, R. Downey, I. Kalimullin, and A. Melnikov. Foundations of online structure theory. *Bull. Symb. Log.*, 25(2):141–181, 2019.

[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$^+$] Nikolay Bazhenov, Matthew Harrison-Trainor, Iskander Kalimullin, Alexander Melnikov, and Keng Meng Ng. Automatic and polynomial-time algebraic structures. *J. Symb. Log.* to appear.

[Bli19] K. V. Blinov. Primitively recursively categorical linear orderings. *Sib. Math. J.*, 60(1):20–26, 2019.

[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.

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

[EG00] Y. Ershov and S. Goncharov. *Constructive models*. Siberian School of Algebra and Logic. Consultants Bureau, New York, 2000.

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

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

[Gon81] S. Goncharov. Groups with a finite number of constructivizations. *Dokl. Akad. Nauk SSSR*, 256(2):269–272, 1981.

[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.

[Khi98] N. Khisamiev. Constructive abelian groups. In *Handbook of recursive mathematics, Vol. 2*, volume 139 of *Stud. Logic Found. Math.*, pages 1177–1231. North-Holland, Amsterdam, 1998.

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

[Kie98] H. A. Kierstead. On line coloring k-colorable graphs. *Israel J. Math.*, 105(1):93–104, 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.

[KMZ] I. Kalimullin, A. Melnikov, and M Zubkov. Punctual degrees and lattice embeddings. *Submitted*.

[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.

[KN08] Bakhadyr Khoussainov and Anil Nerode. Open questions in the theory of automatic structures. *Bull. Eur. Assoc. Theor. Comput. Sci. EATCS*, (94):181–204, 2008.

[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. Israel J.Math, to appear.

[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.

[Rab60] M. Rabin. Computable algebra, general theory and theory of computable fields. *Trans. Amer. Math. Soc.*, 95:341–360, 1960.

[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
Email address: bazhenov@math.nsc.ru

Kazan Federal University
Email address: ikalimul@gmail.com

Massey University & Kazan Federal University
Email address: alexander.g.melnikov@gmail.com

Nanyang Technological University
Email address: selwyn.km.ng@gmail.com