Theoretical Computer Science 674 (2017) 73–98

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Algebraic structures computable without delay ☆

Iskander Kalimullin, Alexander Melnikov *, Keng Meng Ng

ARTICLE INFO

Article history:
Received 27 January 2016
Received in revised form 13 December 2016
Accepted 20 January 2017
Available online 1 March 2017
Communicated by A. Kucera

Keywords:
Polynomial structures
Automatic algebra
Primitive recursion

ABSTRACT

In this article we suggest a new systematic approach to studying algorithms on algebraic structures via primitive recursion. The approach is designed to fill the gap between abstract computable structure theory and feasible (in the sense of polynomial-time, computational or automatic) algebra.

1. Introduction

In the early 1960's, Mal'cev [33] and Rabin [41] independently gave a general definition of an algorithmically presented algebraic structure.

Definition 1.1 *(Mal'cev, Rabin).* A structure with domain ω (natural numbers) is *computable* if its operations and relations are uniformly Turing computable.

If a countably infinite $\mathcal{A}$ is isomorphic to a computable $\mathcal{B}$, then we say that $\mathcal{B}$ is a computable presentation, a computable copy, or a constructivization of $\mathcal{A}$. The notion of a computably presented structure united and extended the earlier definitions of an explicitly presented field [49] and of a "recursively presented" group with a solvable word problem [25].

Much work has been done on computable groups [27,20,10], fields [14,37,36], Boolean algebras [42,19], linear orders [11], computable aspects of model theory [23,35,2,31] and the degree-theoretic properties of algebraic structures [45,50,15]. Investigations of this sort form a field known under the names of *computable structure theory* and *effective algebra*, see books [3,13] and surveys [21]. From a purely technical point of view computable structure theory is more closely related to definability by infinitary formulae [3], $\mathbb{HF}$-definability [14], degree theory [47] and reverse mathematics [44], rather than to any actual computational applications. Nonetheless, computable structures in some natural algebraic classes tend to have computationally "feasible" presentations. We still do not have a satisfactory formal explanation of this phenomenon. Thus we have the following non-trivial question:

When does a computable algebraic structure have a feasible presentation?

☆ We are thankful to Rod Downey and Pavel Alaev for fruitful discussions. The third author is partially supported by the grant MOE-RG26/13. The second author was partially supported by Marsden Fund of New Zealand and Massey Research Fund.

* Corresponding author.
E-mail address: alexander.g.melnikov@gmail.com (A. Melnikov).

http://dx.doi.org/10.1016/j.tcs.2017.01.029
0304-3975/

What does it mean for an infinite algebraic structure to have a feasible presentation? Different branches of effective mathematics suggest different rigorous answers to this question. For example, we could restrict ourselves to algebraic structures that are presented by finite automata [28–30]. Automatic structures have a number of nice properties, including quick computational characteristics and decidability of their first-order theories (and even beyond), but automatic structures tend to be rare. For example, a countably infinite Boolean algebra is finite-automatic iff it is isomorphic to the interval-algebra of the ordinal $\omega \cdot n$, see [30]. Although having a finite-automatic presentation of a structure is highly desirable, it is usually quite difficult to see whether a non-trivial algebraic structure has such a presentation. For instance, using deep results borrowed from additive combinatorics [16], Tsankov [48] has showed that the group of the rationals $(\mathbb{Q}, +)$ cannot be represented by a finite automaton. The result of Tsankov settled a long-standing conjecture of Khoussainov and Nerode (see e.g. [29]). Despite these difficulties, there have been a number of deep works on finite automatic structures [28,30], especially on finite-automatic groups [12,39,4,38].

Cenzer, Remmel, Downey and their co-authors developed a more relaxed and general approach, see survey [7]. More specifically, a computable presentation is *feasible*, or *polynomial time*, if the operations and relations of the structure are polynomial time computable in the length of input. Clearly this definition depends on how we represent the domain ω but we shall not discuss these subtleties here (see [7]). There is a relatively large body of research on polynomial time algebra (e.g., [7,8,6,22]), and some of these results relate computable structure theory with feasible algebra. Nonetheless, there is still a significant gap between these two topics, and deep results relating computable structure theory and polynomial time algebra are rare. In this paper we suggest a *systematic* approach designed to fill this gap.

1.1. From computable to feasible

When considering computable structures we allow algorithms to be extremely inefficient. For example, we may use an unbounded search through ω as long as we can *prove* that it will halt. More formally, our algorithms do not even have to be *primitive recursive*. Nonetheless, in several common algebraic classes we can show that every computable structure has a polynomial-time computable copy. These classes include linear orders [22], broad subclasses of Boolean algebras [5], some commutative groups [8,6], and other structures [7]. Interestingly, many known proofs of this sort (e.g., [7,8,6,22]) are essentially focused on making the operations and relations on the structure primitive recursive, and then observing that we get a polynomial-time presentation almost for free. It appears that primitive recursion plays a rather important intermediate role in such proofs. This thesis is also supported by a number of *negative* results in the literature. Indeed, to illustrate that a structure has no polynomial time computable copy, it is sometimes easiest to argue that it does not even have a copy with primitive recursive operations, see e.g. [8]. For this technical reason Cenzer and Remmel [7] came up with the following general definition.

Definition 1.2. An algebraic structure is *primitive recursive* if its domain is a primitive recursive subset of ω and the operations and relations of the structure are (uniformly) primitive recursive.

Our initial thought was that primitive recursive structures would be an excellent candidate for an intermediate class between computable structures and feasible structures. However, we very soon realized that the above definition is a bit too relaxed. In a primitive recursive structure, we may see new elements appearing in the structure extremely slowly; the principal function of the domain might not be primitive recursively bounded. This feature can be exploited to show that most computable structures have primitive recursive copies. For example, as observed by Alaev (personal communication), every computable structure whose finitely generated substructures are finite has a primitive recursive copy. Indeed, we can simply keep elements of ω out of the domain until we wait for a larger finite substructure to be revealed in the computable copy. In particular, any computable relational structure in a finite language admits a primitive recursive copy.[1] This fact strongly suggests that primitive recursive structures are not "truly" primitive recursive, i.e. they seem too close to (general) computable structures to be a good intermediate notion.

We suggest that a truly "non-delayable" computable presentation must minimally satisfy the following definition:

Definition 1.3. A countable structure is fully primitive recursive (fpr) if its domain is ω and the operations and predicates of the structure are (uniformly) primitive recursive. We also fix the convention that all finite structures are fully primitive recursive by allowing the domain to be a finite initial segment of ω.

The reader should note that the situation here is quite different from computable structures where the domain can typically be assumed an arbitrary computable subset of ω. Indeed, a fully primitive recursive structure must reveal itself without any unbounded delay. One of our main results (Theorem 3.2, to be stated) combined with the observation of Alaev discussed above imply:

Fact 1.4. There exist primitive recursive structures that have no fully primitive recursive presentation.

[1] As noted by the anonymous referee, this observation was known to Remmel and Nerode long before Alaev.

Although primitive recursive functions can be computationally very inefficient, we will see that, when compared with computable structures, fpr structures behave much more impatiently and enumerate themselves more rapidly. Thus, we may informally call such structures *computable without delay*.

We are unaware of any previous systematic study of fpr structures in their own right. We note that primitive recursive presentations upon ω have been used by Cenzer and Remmel (at least once) as a technical tool for showing that certain structures have no feasible copy (see [8]). We also note that Alaev has recently and independently suggested an alternate approach to computable structures via primitive recursion [1].

1.2. The main questions

We are ready to state the first main question addressed in the paper:

When does a computable structure have a fully primitive recursive presentation?

The natural morphisms in the category PR_ω of fully primitive recursive structures are primitive recursive isomorphisms with primitive recursive inverse. We call such isomorphisms *fully primitive recursive* (fpr). For example, the dense linear order $(\mathbb{Q}, <)$ clearly has a fpr presentation. It is also clear that any two such presentations are computably isomorphic. Nonetheless, it is not hard to see that there exist two (in fact, infinitely many) fpr-copies of $(\mathbb{Q}, <)$ that are not fpr-isomorphic. We note that there has been a lot of work on the number of (Turing) computable presentations of algebras up to computable isomorphism [31,13,18,46,32,40,26], but not much is known about the situation with primitive recursive isomorphisms. Given a fpr algebraic structure $\mathcal{A}$, we ask the second main question of the paper:

How many fpr presentations does $\mathcal{A}$ have up to fpr isomorphism?

The rest of the paper is devoted to a systematic study of the above two questions and comparing them with the respective problems in computable structure theory [3,13]. We will also address some other problems, but most of these directions we leave wide open. Although we conjecture that most fpr structures that appear in this paper have polynomial-time copies, verifying this claim is outside the scope of this paper.

We now turn to a detailed discussion of the results.

1.3. Existence of a fpr presentation

We open the paper with a rather satisfying positive result.

Theorem 1.5. *In each of the following classes, every computable structure has a fully primitive recursive presentation:*

(1) Equivalence structures.
(2) Linear orders.
(3) Torsion-free abelian groups.
(4) Boolean algebras.
(5) Abelian p-groups.

We note that (1) follows from an observation of Cenzer and Remmel [7], and (2) easily follows from Grigorieff [22]. We outline the proofs of (1) and (2) for the sake of completeness of exposition. The idea behind (3) was known to Downey[2] in a different set-up, but it has never been published. Parts (4) and (5) are new (but [5] contains several related partial results). Our proofs exploit techniques specific to each class under consideration. For example, our proof of (5) uses a 70-year-old theorem of Prüfer, and the proof of (4) exploits the old theorem of Remmel [43] and presentations by trees [19] blended within a priority construction. Although there are certain similarities between the proofs of the different parts of Theorem 1.5, we could not come up with any convenient sufficient condition for a computable structure to have a fpr presentation. We leave the existence of such a condition as an open question.

In contrast to Theorem 1.5, Theorem 3.2 says that *in each of the following classes, there exists a computable structure that does not admit a fpr presentation:*

(1) *torsion abelian groups,*
(2) *Archimedean ordered abelian groups,*
(3) *undirected graphs.*

[2] Personal communication with the second author.

Part (1) of Theorem 3.2 was proved in [8], we will outline a rather elementary proof. Parts (2) and (3) are new. Observe that Theorem 3.2(1) contrasts with (3) and (5) of Theorem 1.5. As we will discuss in Subsection 3.2, (2) of Theorem 3.2 is quite unexpected since both torsion-free abelian groups (Theorem 1.5(3)) and Archimedean ordered abelian groups have computable copies with a computable basis [40,20,24], and the proof of Theorem 1.5(3) will be heavily exploiting this property.

Recall that every computable graph has a primitive recursive copy (not upon ω). Furthermore every computable locally finite graph has a fully primitive recursive presentation (as one can easily verify this, we omit details). It is thus natural to conjecture that *every* computable graph has a fpr presentation. Part (3) of Theorem 1.5 refutes this conjecture. The proof, while not difficult, does employ some novel ideas and may be of technical interest to the reader.

We conclude that, in contrast to primitive recursive structures, *fully* primitive recursive structures behave quite differently from computable structures. We feel that this justifies further investigations into this notion. In Subsection 3.3 we also suggest and briefly discuss a primitive recursive analogy of 1-decidability. This approach can be further extended to n-decidability or decidability, although we do not do this in the paper. This stronger notion is designed to eliminate several unsatisfactory features that fpr structures may exhibit in some classes.

1.4. Uniqueness of a fpr presentation

Following Mal'cev, we say that a structure is *computably categorical* or *autostable* if it has a unique computable copy up to computable isomorphism. Note that the inverse of a primitive recursive function does not have to be primitive recursive. Thus it is reasonable to define an isomorphism f to be *fully primitive recursive* (fpr) if f and f^{-1} are both primitive recursive. Fully primitive recursive isomorphisms preserve all properties of fpr structures at the right (primitive recursive) level, and thus such isomorphisms seem to be the most natural morphisms between fpr structures.

Definition 1.6. A fully primitive recursive structure $\mathcal{A}$ is *fpr-categorical* if it has a unique fully primitive recursive presentation up to fully primitive recursive isomorphism.

We were able to characterize fpr-categorical structures in many common classes.

Theorem 1.7.

(1) An equivalence structure S is fpr-categorical iff it is either of the form $F \cup E$, where F is finite and E has only classes of size 1*, or S has finitely many classes at most one of which is infinite.*
(2) A linear order is fpr-categorical iff it is finite.
(3) A Boolean algebra is fpr-categorical iff it is finite.
(4) An abelian p-group is fpr-categorical iff it has the form $F \oplus \mathbb{V}$, where $p\mathbb{V} = \mathbf{0}$ and F is finite.
(5) A torsion-free abelian group is fpr-categorical iff it is the trivial group $\mathbf{0}$*.*

Even though Theorem 1.5 typically produces the most "boring" fpr presentations in each class, Theorem 1.7 says that almost all structures in these classes have complex ("irregular", "unpredictable") fpr presentations. In fact, in many cases we can even diagonalize against all *computable* isomorphisms between two fpr copies of a structure. We also note that Theorem 1.7 resembles the following result of Khoussainov and Nerode [28]: A structure is automatically categorical iff it is finite.

According to our definition, every fpr-categorical structure must have a fully primitive recursive (thus, computable) copy. Theorem 1.7 suggests that fpr-categorical structures in common classes are necessarily computably categorical and tend to be trivial. Nonetheless, in Proposition 4.2 we will construct the first example of a fpr-categorical structure that is not trivial (in the sense that will become clear later). Furthermore, to our surprise, fpr-categorical structures do not form a proper subclass of computably categorical structures.

Theorem 1.8. *There exists a fpr-categorical structure which is not computably categorical.*

The proof of Theorem 1.8 combines several novel strategies and is quite combinatorially involved. We also note that the structure witnessing Theorem 1.8 is rigid and is in a finite language consisting of four unary function symbols. We leave open whether such structures can be found in the common algebraic classes (e.g., groups or fields), and we conjecture that with some effort such examples can be constructed.

1.5. Further topics

There are many interesting questions one can ask about fpr structures, but in this paper we touch only a few further subjects. First of all, we note that Theorem 1.8 has several pleasant consequences. For instance, the structure $\mathcal{A}$ witnessing Theorem 1.8 cannot be uniformly fpr-categorical (meaning that fpr ismorphism cannot be witnessed by a pair of primitive

recursive functionals). Indeed, otherwise we could produce a c.e. Scott family consisting of first-order $\exists$-formulae, thus giving that the structure is (relatively) computably categorical. This observation explains why examples of non-computably categorical, fpr-categorical structures are hard to find in the natural non-universal classes where relative and plain computable categoricity tend to be the same.

Also, as we will show in Proposition 4.6, the proof of Theorem 1.8 can be modified to show that there exists a fpr structure $\mathcal{A}$ and a relation P on $\mathcal{A}$ which is primitive recursive in all fpr copies of $\mathcal{A}$, but there is a computable presentation of $\mathcal{A}$ in which P is not even computable.

Finally, we strongly conjecture that the proof of Theorem 1.7 can be modified to show that in all these classes we have one or infinitely many fpr presentations up to fully primitive recursive isomorphism. We leave open whether there exists a fpr structure with exactly two (or $0 < n < \infty$) fpr presentations up to fully primitive recursive isomorphism.

2. Primitive recursion

Before we turn to a more detailed discussion of the new notion, we should remind the reader what primitive recursive functions are, and how to deal with them.

Recall that a recursive function is *primitive recursive* if the function can be generated from the basic functions $s(x) = x+1$, $o(x) = 0$, $I^n_m(x_1, \dots, x_n) = x_m$ by composition and the primitive recursion operator $h = \mathcal{P}(f, g)$:

$$h(x_1, \dots, x_n, 0) = f(x_1, \dots, x_n),$$
$$h(x_1, \dots, x_n, y+1) = g(x_1, \dots, x_n, y, h(x_1, \dots, x_n, y)).$$

We note that the history of primitive recursive functions goes back all the way to Dedekind [9] in the 19th century. The unbounded search operator (the μ-operator) cannot be used in the scheme but (as it is well-known) we may allow the bounded μ-operator.

Primitive recursive functions are much easier to understand if one follows the restricted Church–Turing thesis for primitive recursive functions. More specifically, if we can describe our algorithm without using unbounded loops (such as WHILE ... DO, REPEAT ... UNTIL, and GOTO in a Pascal-like language) then our function is primitive recursive. Even less formally, if our algorithm does not have instructions of the form "wait until some effective process halts", then our algorithm will be primitive recursive.

It will also be important that all primitive recursive functions are total.

3. Existence of fpr presentations

3.1. Proof of Theorem 1.5

(1) If the given computable equivalence structure has finitely many classes, we can clearly produce a fpr presentation by first fixing finitely many parameters. Otherwise, assume there are infinitely many distinct classes in E. We build a fpr presentation I of E, and a computable isomorphism $f : I \mapsto E$. At stage s of the construction we check if the next element x of E has been decided. That is, we check if $E(x, y)[s]\downarrow$ for every $y < x$. If x is not yet decided then at stage s we introduce a new element into $I[s]$ and declare it unrelated to all existing elements of I. Otherwise x is decided at stage s. If x is E-unrelated to all $y < x$ then we pick the least $n \notin dom(f)$ and define $f(n)\downarrow = x$. Otherwise $E(x, y)$ holds for some collection of $y < x$, where $y \in Rng(f)$. Introduce a new element n in $I[s]$ and declare it to be I-related to $f^{-1}(y)$ (and taking the transitive closure if necessary). Declare $f(n) = x$.

Clearly f is total because E has infinitely many distinct classes. Thus f witnesses that $E \cong I$. Note that f is computable but not necessarily primitive recursive. I is fpr because at every stage s, $I(x, y)[s]$ is decided for every $x, y < s$.

(2) The proof then splits into the following cases.

Case 1: $\mathcal{L}$ has a left limit point b. That is, $\mathcal{L}$ contains an element b with no predecessor and where b is not the least element of $\mathcal{L}$. We describe how to produce a fpr presentation $\mathcal{A}$ and a computable isomorphism $f : \mathcal{A} \mapsto \mathcal{L}$. In $\mathcal{A}$ we fix $a = f^{-1}(b)$ and grow an increasing sequence of elements $a_0 < a_1 < a_2 < \dots < a$. At the beginning we start with $a_0 \in \mathcal{A}$ and define $f(a_0) < b$ to be the first enumerated in $\mathcal{L}$. While waiting for the next element to be enumerated in $\mathcal{L}$ we continue to place the elements of the sequence $a_0 < a_1 < a_2 < \cdots$ in $\mathcal{A}$.

Suppose at some stage s we find a new element x enumerated in $\mathcal{L}$. If $x > b$ then we place a corresponding element $n > a$ in $\mathcal{A}$ and set $f(n) = x$. Otherwise $x < b$ and assume that i is the largest so far such that $f(a_i)\downarrow$. By the construction we assume that $f(a_i)$ is the largest element $< b$ present in $\mathcal{L}$ (except possibly for x), and that for every $y < f(a_i)$ in $\mathcal{L}$, we have already defined $f^{-1}(y)$. Now if $x < f(a_i)$ in $\mathcal{L}$ then we introduce a corresponding element n in $\mathcal{A}$ and declare $f(n) = x$. Place n in $\mathcal{L}$ appropriately. On the other hand if $x > f(a_i)$ then we declare $f(a_{i+1}) = x$.

Clearly, f is a partial isomorphism at every stage, and is clearly surjective as each new $x \in \mathcal{L}$ is immediately put in the range of f. Furthermore f is total by the assumption that b is a left limit point of $\mathcal{L}$. Clearly $\mathcal{A}$ is fpr.

Case 2: $\mathcal{L}$ has a right limit point. Same as Case 1.

Case 3: $\mathcal{L}$ contains a sub-interval of order-type $\omega+\omega^*$. Then we (non-uniformly) fix the end-points $a<b$ of the interval $\omega+\omega^*$. We build a copy $\mathcal{A}\cong\mathcal{L}$ and a computable isomorphism $f:\mathcal{A}-\left[f^{-1}(a),f^{-1}(b)\right]\mapsto\mathcal{L}-[a,b]$. The strategy to build $\mathcal{A}$ is simple. Monitor $\mathcal{L}-[a,b]$, and each time a new element shows up in $\mathcal{L}-[a,b]$ we place a corresponding element in $\mathcal{A}-\left[f^{-1}(a),f^{-1}(b)\right]$ and extend f. While waiting we simply grow the $\omega+\omega^*$ chain in $\mathcal{A}$.

Case 4: Assume that none of the above holds. We may assume that $\mathcal{L}$ has a greatest element, because otherwise we can proceed as in Case 1 for $\mathcal{L}\cup\{\infty\}$. Similarly $\mathcal{L}$ has a least element. Since the greatest and least elements of $\mathcal{L}$ are not limit points, we can continue to argue that $\mathcal{L}=\omega+\mathcal{L}'+\omega^*$ for some $\mathcal{L}'$. In fact $\mathcal{L}'$ is non-empty as $\omega+\omega^*$ is not a sub-interval of $\mathcal{L}$. Any $x\in\mathcal{L}'$ must have a successor and a predecessor (otherwise we are in Case 1 or Case 2). So any $x\in\mathcal{L}'$ is part of a $\mathbb{Z}$-chain. As $\omega+\omega^*$ is not a suborder of $\mathcal{L}$ we conclude that the $\mathbb{Z}$-chains of $\mathcal{L}'$ are dense with no greatest and no least $\mathbb{Z}$-chain. In other words, $\mathcal{L}$ is of the form $\omega+\mathbb{Z}\cdot\eta+\omega^*$. This ordering clearly has a fpr presentation.

(3) Recall that elements $a_1,\ldots,a_k$ of an abelian group are linearly independent if they are linearly independent over $\mathbb{Z}$, i.e., if $n_1a_1+\ldots+n_ka_k=0$ implies $n_1=\ldots=n_k=0$ for any integer coefficients $n_1,\ldots,n_k$. A basis of a group is a maximal linearly independent subset of the group. The rank of an abelian group, which is the cardinality of its basis, is an invariant of the group. A basis should not be confused with a generating set; for example, $\{(2,0),(2,4)\}$ is a basis of $\mathbb{Z}^2$, but it does not generate the group under $+$ and $-$. On the other hand, each generating set of a free abelian group can be replaced by a linearly independent one, and we will assume all generating sets of free abelian groups under consideration are linearly independent. We will also be using the well-known fact that a finitely generated subgroup of a torsion-free abelian group is free abelian.

It is well-known that every computable torsion-free abelian group has a computable copy with a computable basis [40] (see also [24] for a modern proof). We explain the case when the rank of the group is infinite, the case of any finite rank is simpler (just remove the basis-extension strategy from the construction below).

Thus, without loss of generality we may assume that G has an infinite computable basis $(a_i)_{i\in\omega}$. The idea is to keep building the subgroup generated by the a_i (which is isomorphic to the free abelian group of rank ω) while we wait for another generator of G to show up.

We are building a fpr $H\cong G$. Suppose at a stage s we have enumerated a finite partial group H_s with a basis $b_1,\ldots,b_s$ and a generating set $e_1,\ldots,e_s$. We also assume that we have defined a partial embedding $\psi_s:H_s\to G_s$ such that $range(\psi_s)\subseteq G_s$ and $\phi_s(b_i)=a_i$ for some $i\le s$ such that a_i have already been seen at this stage (ϕ_s will not be primitive recursive). Since each b_i is the intended isomorphic pre-image of a_i, every element h of H_s is uniquely associated with a (reduced) linear combination of $b_0,\ldots,b_s$:

$$mh=\sum_{i\le s}n_ib_i,$$

where $m,n_0,\ldots,n_s\in\mathbb{Z}$ and $m\neq0$. The partial group H_s always comes together with its *generating set* $e_0,\ldots,e_s$ that makes it a (partial) free abelian group upon these generators:

$$H_s=\langle e_0\rangle_m\oplus\ldots\oplus\langle e_s\rangle_m,$$

where $\langle e_i\rangle_m=\{-me_i,\ldots,-e_i,0,e_i,\ldots,me_i\}$ and m is some positive integer (which will be determined by the construction). Such a generating set exists because any finitely generated torsion-free abelian group is free abelian. Note that the number of elements in the generating set should be the same as in the basis $b_1,\ldots,b_s$, since the rank of the free $\mathbb{Z}$-span of H_s is its invariant.

There are three types of strategies that work together towards building a fpr presentation of G.

The basis-extension strategy

If the strategy becomes active at stage s, it introduces a new element b_{s+1}, and initiates the enumeration of the free group naturally extending H_s to $H_s\oplus\langle b_{s+1}\rangle$. In the notation as above, we set $e_{s+1}=b_{s+1}$ and keep the rest of the e_i $(i\le s)$ unchanged.

Note that all actions of the strategy are primitive recursive. In the construction, we will also wait for a_{s+1} to show up in the basis, and then we will set $\phi_{s+1}(b_{s+1})=a_{s+1}$. Then ϕ is perhaps not primitive recursive, but the construction will be.

The copy-delay strategy

This strategy is always given (as its input) a finitely generated partial (free) abelian group H_s upon a fixed generating set $e_1,\ldots,e_s$. The strategy then makes s steps towards extending H_s naturally to the corresponding free abelian group $\bigoplus_{i\le s}\langle e_i\rangle$.

Finally, we need to make sure that ϕ_s is an isomorphism onto. The strategy below ensures that all elements in G eventually get ϕ-preimages. The strategy itself is not primitive, but it will be stretched in the construction into a sequence of primitive actions, with copy-delay and basis-extension strategies acting in-between.

The onto-strategy

The strategy has a witness $g \in G$, and it waits for a reduced linear combination

$$mg = \sum_{i \le s} n_i a_i$$

to show up in G. The strategy then decides whether the free group naturally extending the current H_s (in the sense of the copy-delay strategy) will ever have an element h' with the property $mh' = \sum_{i \le s} n_i b_i$.

Remark. Note that the latter is decidable. Recall that H_s comes together with its generating set $e_1, \dots, e_s$. Replace the b_i in the sum by the respective linear combination of the e_j (say, $b_j = \sum_i m_{i,j} e_i$) and see whether the resulting linear combination $\sum_j k_j e_j$ satisfies $m | k_j$ for all j.

If such an h' exists, then wait for it to appear in the enumeration of the free group extending H_s, and define $\phi(h') = g$. Otherwise, introduce a *new* element h' with the desired property. Then find a new set of generators $e'_1, \dots, e'_s$ in the new extended partial group (to be used by the copy-delay strategy) extending the partial group further if necessary. Finally, define $\phi(h') = g$ and declare that the action of the strategy is finished.

Construction

Let the strategies act one after another according to some natural order, but with one important restriction. More specifically, we will let the copy-delay and the basis-extension strategy alternate their actions while the onto-strategy waits for its computation to be finished. We will also postpone the definition of ϕ in the basis-extension strategy by allowing $\phi(b_s)$ to be decided later in the construction. Also, whenever we introduce a new element into H, we always use the least element of ω never used to index elements of the group H that we construct.

Verification

It is clear that $\phi = \cup_s \phi_s$ is a homomorphism of H onto G. Since every element of G corresponds to a linear combination of $(a_i)_{i \in \omega}$, the onto-strategy ensures that the homomorphism ϕ is surjective. Furthermore, ϕ maps a basis of H to a basis of G, thus it is actually injective and hence an isomorphism.

We now check that H is fully primitive recursive. Very informally, we did not use any unbounded delay in the definition of the operations on $H = \bigcup_s H_s$, thus it is fpr. More formally, we need to verify that the group operations $+$ and $-$ are primitive recursive. Recall the domain of H is ω. If $h, h' \in H$ then we can primitively recursively find an s such that both $h, h' \in H_s$ (indeed, we may arrange the construction so that, say, $s < \max\{h, h'\}$). Again primitively recursively, we can find generators $e_1, \dots, e_s$ of H_s and express h, h' as linear combinations of these generators. Since we very often have copy-delay stages acting, we can primitively recursively find the linear combination of the e_j corresponding to $h + h'$, and thus a t and the element h'' of $H_t \supseteq H_s$ representing this linear combination. Then evidently $h + h' = h''$. Similarly, $-$ is primitive recursive as well. Indeed, for every h if $h \in H_s$ then $-h \in H_s$.

(4) Suppose that $\mathcal{B}$ is a computable Boolean algebra. We write $(b)_{\mathcal{B}}$ or simply (b) for the ideal of $\mathcal{B}$ generated by $b \in \mathcal{B}$. We will also use the standard terminology (such as atom, atomless etc.), see [19] for these definitions. For instance, we will be using the standard partial ordering induced by the operations on $\mathcal{B}$. Under this order, $x \le y$ (x is below y) iff $x \in (y)$.

If $\mathcal{B}$ is finite, then there is nothing to prove. Without loss of generality, we may also assume that $\mathcal{B}$ has no atomless elements. Indeed, if $\mathcal{B}$ had an atomless $b \in \mathcal{B}$, then $\mathcal{B} = \mathcal{A} \oplus (b)$ and we could just keep building the atomless (b) of $\mathcal{B}$ bounded by b while waiting for new elements to show up in $\mathcal{B}$. A rather routine argument shows that this way we'll end up with a fpr presentation of $\mathcal{B}$, we leave reconstructing the elementary formal details to the reader (see also [5] for a similar proof).

We concentrate on the more interesting case when $\mathcal{B}$ is infinite and has no atomless elements. We thus we assume that $\mathcal{B}$ is infinite and *atomic*; that is, every element of $\mathcal{B}$ bounds an atom. The main idea behind the proof can be informally described as follows. While we wait for another element to split in the computable Boolean algebra, we insert more atoms into our fpr presentation. Since there will be plenty of extra atoms, we will need to somehow find their isomorphic images in the computable Boolean algebra that we copy. To do this we will use a priority construction and some techniques standard for computable Boolean algebras (to be explained).

We are building a fpr presentation $\mathcal{P}$ of $\mathcal{B}$ and a Δ^0_2 map

$$\phi : \mathcal{P} \to \mathcal{B},$$

such that ϕ is "almost" an isomorphism of $\mathcal{P}$ onto $\mathcal{B}$. That is, we relax the definition of an isomorphism and allow the ϕ-preimage of an atom in $\mathcal{B}$ to be the sum of finitely many atoms in $\mathcal{P}$. It is well-known that any two Boolean algebras almost isomorphic in this sense are in fact isomorphic [43]. Since at the end ϕ will be merely Δ^0_2, at every stage we can define only our best guess on $\phi(x)$, and this may later change. We will then argue that there will be at most finitely many changes of our definition of $\phi(x)$ for every x.

We may assume that in the effective enumeration $(\mathcal{B}_s)_{s \in \omega}$ of $\mathcal{B}$ either exactly one atom of $\mathcal{B}_{s-1}$ splits in $\mathcal{B}_s$ into two atoms, or $\mathcal{B}_{s-1} = \mathcal{B}_s$. If c is the atom in $\mathcal{B}_{s-1}$ that splits in $\mathcal{B}_s$ then we say that c *s-splits*, and we also say that the stage

s at which the element was split is a *splitting stage* of c, or simply a splitting stage if knowing c is not necessary. It will be convenient to use a c.e. binary generating tree [19] to represent $\mathcal{B}$. In such a subtree of $2^{<\omega}$, the root represents $\mathbf{1}$, nodes represent generators of a Boolean algebra, and the join of two nodes corresponds to the union (supremum) of two generators. See [19] for an excellent exposition. Without loss of generality, we may assume that we are given such a computably enumerable subtree of $2^{<\omega}$ of generators in $\mathcal{B}$. Since $\mathcal{B}$ is atomic, every node in the tree bounds a terminal node. We identify the nodes on the tree with the respective generators of $\mathcal{B}$.

The *basic copying strategy* can be described as follows. Suppose an element $x \in \mathcal{B}$ s-splits, say

$$x = x_1 \oplus x_2.$$

Then we pick $y \in \mathcal{P}$ such that $\phi(y) = x$ and such that y is not in any *queue* (to be defined) and split y into the (disjoint) sum of y_1 and y_2. We postpone the definition of ϕ on y_1 and y_2. At every stage $t > s$ that is not a splitting stage of any element, we will introduce more atoms below y by further splitting y_1 and y_2. Keep splitting until the next splitting stage s' is found. Depending on whether this new split occurs within (x) or its complement, at stage s' the algebra (x) has $k = 3$ or $k = 2$ atoms, respectively. Then pick any k atoms in (y) and map them into the k atoms in (x) under ϕ, but do not define ϕ on the rest of the atoms $z_0, \dots, z_n$ in (y). These remaining atoms (if there are any) now form the *queue of x-followers* that will be assigned to its own *isomorphism-builder* strategy (to be introduced) whose task will be to find a stable definition of ϕ on these remaining atoms.

The isomorphism-builder strategy

The strategy will be associated with its queue $z_0, z_1, \dots, z_n$ of x-followers. The strategy picks the least (under the natural order on ω) element a below x that currently looks like an atom, i.e., is an atom in $\mathcal{B}_s$. Call a the witness of the strategy. Once such an a is found, define $\phi(z_i) = a$ for all $i \le n$. On the remaining generators below y, set ϕ to be any isomorphism of $(y - \bigoplus_{i \le n} z_i)$ onto (x). (Here $u - v$ stands for the complement of $v \wedge u$ in (u).)

Remark

We will argue that the number of atoms in the finite Boolean algebra (x) will never exceed the number of atoms in (y) at every stage. Furthermore, we will see that the extra atoms (if there are any) are exactly the $z_0, z_1, \dots, z_n$ forming the queue of the strategy. Thus, the definition of ϕ is consistent.

Priority ordering

The copying strategies have no priority and cannot be initialized. We assign each isomorphism-builder strategy a certain priority in the construction; this priority depends on the stage at which the respective queue was introduced (the earlier it was formed, the higher priority it receives). An isomorphism-builder strategy can be initialized, and it may also modify its queue (see below).

Initialization and queue modification

As we noted above, only an isomorphism-builder strategy can be initialized. Suppose we have two isomorphism-builder strategies I' and I whose queues follow x' and x, respectively, where x' is below x. As we will see, I' must have a weaker priority than I. If I discovers that its witness is not an atom, we initialize I' and adjoin its queue to the queue of I. This is done before I chooses a new witness. The initialized strategy will never act again.

Construction

At every splitting stage we let the isomorphism-builder strategies re-define ϕ if necessary. As explained above, if an isomorphism-builder strategy I whose queue follows x is forced to change its witness, then we first initialize all weaker priority strategies working below x and only then we let the strategy I act. We let the copying strategies act according to their instructions between splitting stages.

Verification

First, we argue that every isomorphism-builder strategy can successfully define ϕ as described in its basic module above. This follows by induction. At the beginning, immediately after the queue $z_0, z_1, \dots, z_n$ of x-followers is introduced, the respective strategy I guesses that one of the at most three atoms of $\mathcal{B}_s$ below x, say a, is an atom in $\mathcal{B}$. Then the strategy defines ϕ within (y) so that $\phi(z_j) = a$ for all j, and $\psi : (y - \bigoplus_{j \le n} z_j) \to (x)$ is an isomorphism. The number of atoms in the queue was initially chosen so that the Boolean algebras $(y - \bigoplus_{j \le n} z_j)$ and (x) have equal number of atoms, see the basic module of the copying strategy. But then the strategy can define ϕ according to its instructions.

Now suppose the strategy I has already acted several times, and without loss of generality there will be at most finitely many other isomorphism-builder strategies $I', I'', \dots$ whose queues follow elements $x', x'', \dots$ below x. Since these strategies have been introduced after the queue of x was formed, all these strategies $I', I'', \dots$ have weaker priorities than I. The only reason I needs to act again is that its previous witness has split. But this means that all the strategies $I', I'', \dots$ must be initialized and must immediately adjoin their queues to the queue of I. After these extra elements have been adjoined, the strategy I acts. By a straightforward induction, the finite Boolean algebra (y) will have more atoms than (x). Furthermore,

we claim that these extra atoms are exactly the elements in the new extended queue $z_0, z_1, \ldots, z_m$ of I. Indeed, the only reason we introduce any extra atoms to (y) (when compared with (x)) is because of the actions of various copying strategies working below y. In other words, the queue of I has accumulated all the extra atoms that we have put so far below y, when compared with x. Thus, the strategy may safely define $\phi(z_j) = a$, where a is as above, and then define ϕ to be any isomorphism of $(y - \bigoplus_{i \le m} z_i)$ onto (x).

Note that, since the algebra is atomic, every isomorphism-builder strategy that has ever been formed eventually is either initialized or finds a true atom below its respective x. If an isomorphism-builder strategy is initialized, then by induction we have that all atoms in its queue eventually receive a stable ϕ-definition. The base of induction follows from the nature of the search being Δ^0_2, and from the analysis in previous paragraph. We conclude that any extra atom z of $\mathcal{P}$ that was introduced by copying strategies will eventually find itself in a queue that follows some x, and so that the witness a of the respective isomorphism-builder strategy is a true atom in $\mathcal{B}$.

We may replace the computable Boolean algebra $\mathcal{B}$ by its computable generating tree, with T_s naturally representing the finite $\mathcal{B}_s$. Now splitting an element $x \in \mathcal{B}_{s-1}$ in $\mathcal{B}_s$ corresponds introducing two children of the respective terminal node of T_{s-1}. It will be most convenient to assume that, together with $\mathcal{P}$, we are building a generating tree Y of $\mathcal{P}$. The tree that we build tries to "copy" the tree T of $\mathcal{B}$, in the following sense. Re-defining ψ below y corresponds to replacing the current finite subtree (y) by another finite binary tree with the same number of leaves, and whose leaves are labeled by the same (current) atoms in (y). The new tree will be the same as the subtree of T_s rooted in x, with the exception that the subtree with leaves $z_0, \ldots, z_m$ will correspond to a single atom a below x. Then showing that ψ is total and Δ^0_2 is the same as arguing this process of re-arranging Y must stabilize for every node of the tree Y.

The isomorphism-builder strategies that are introduced late in the construction will be following elements x far from the root of T, and thus by induction y will be far from the root of Y, where $\psi(y) = x$. Recall that an isomorphism builder strategy (unless initialized) may possibly rearrange ψ only below some fixed y corresponding to x. Furthermore, the basic module of the isomorphism-builder strategy ensures that, once a stable atom a below its x is found, ϕ will never be re-defined below x up to, and including, the elements of $\mathcal{P}$ that were used by the strategy in its last action. (Those were the atoms $z_0, \ldots, z_m$ and a few more atoms corresponding to the part of (x) seen at that stage.) It means that in the tree of $\mathcal{P}$ each level can be re-arranged at most finitely many times. Thus, ψ is indeed total and Δ^0_2.

But this means not only that every element of $\mathcal{P}$ eventually receives a stable definition of ψ, but also that ψ is an almost-isomorphism. Indeed, it is an almost-isomorphism on every subalgebra $\mathcal{P}_s$, where $\mathcal{P} = \bigcup_s \mathcal{P}_s$ is defined by the construction. To see why, note that each $\mathcal{P}_s$ is generated by Y_s. But ψ is eventually stable on Y_s, and thus on $\mathcal{P}_s$ as well. Also, ψ is an almost-isomorphism of Boolean algebras (in the sense as above), by its very definition. Therefore, as noted above, $\mathcal{P} \cong \mathcal{B}$.

It remains to observe that every two elements k, l of $\mathcal{P}$ are contained in a finite subalgebra $\mathcal{P}_s$, where s (and thus $\mathcal{P}_s$) can be reconstructed in a primitive recursive way given k, l. We then primitively recursively compute the standard operations in $\mathcal{P}_s$. Therefore, $\mathcal{P}$ is fully primitive recursive.

(5) Somewhat unexpectedly, this case is not very much different from the case of equivalence structures (see (1) above), but it requires a less straightforward preliminary analysis and applications of basic abelian group theory. We assume that the reader is familiar with the classification of finitely generated abelian groups and with the standard terminology of abelian group theory that can be found in any standard text.

Suppose we are given a computable abelian p-group $\mathcal{G}$. We aim to build a fpr presentation $\mathcal{I}$. Recall that the *socle* of $\mathcal{G}$ is the $\mathbb{Z}_p$-vector space

$$\mathcal{G}[p] = \{a \in \mathcal{G} : pa = 0\}.$$

If the socle of $\mathcal{G}$ is finite, then we claim that $\mathcal{G}$ has to be isomorphic to a group of the form $F \oplus \bigoplus_{i \le k} \mathbb{Z}_{p^\infty}$ where F is finite. In this case we can clearly produce a fpr presentation of $\mathcal{G}$. We now explain why the above claim is indeed true. Consider the following possibilities. If the divisible part had infinite rank then the socle would clearly be infinite. Now if the reduced part had at least one non-zero element h of infinite height (that is, $\forall k \exists b_k\, p^k b_k = h$), then the socle would have infinite dimension, as in this case there must exist infinitely many pairs (k, k') such that the respective elements $p^{k-1}b_k - p^{k'-1}b_{k'} \in \mathcal{G}[p]$ are independent over $\mathbb{Z}_p$. So the reduced part must contain only elements of finite height, and therefore by a well-known theorem of Prüfer [17] the reduced part splits into a direct sum of cyclic p-groups. If there are infinitely many summands in this sum, then the socle is again infinite. We conclude that the only possibility is when $\mathcal{G} \cong F \oplus \bigoplus_{i \le k} \mathbb{Z}_{p^\infty}$, where F is finite. A group of this from clearly has a fpr presentation.

Let us therefore assume that the socle of $\mathcal{G}$ is infinite. As in (1), the socle will be used by the delaying strategy when building a fpr presentation $\mathcal{I}$ of $\mathcal{G}$. We enumerate a fpr presentation of the infinite-dimensional vector space $\mathbb{V}_p$ over the finite field $\mathbb{Z}_p$. The plan is to build $\mathcal{I}$ "around" $\mathbb{V}_p$.

At stage s we have a finite group $\mathcal{I}_s = H_s \oplus \bigoplus_{i \le k_s} \mathbb{Z}_p$ on an initial segment of ω, an isomorphic embedding $\psi_s : H_s \to \mathcal{G}$ that is perhaps not yet defined on $\bigoplus_{i \le k_s} \mathbb{Z}_p$. The subgroup $\bigoplus_{i \le k_s} \mathbb{Z}_p$ will be the part of $\mathbb{V}_p$ enumerated so far which has not yet been put into H_s. There are two basic strategies that will be acting together towards building $\mathcal{I}$.

The copy-total strategy

The strategy has a witness, an element $v \notin H_s$ of the fixed basis of the currently built part of $\mathbb{V}_p$. Its main task is to find a ψ-image for v. This strategy is not primitive recursive, and its actions will be stretched using the delay strategy (to be

introduced below). The copy-total strategy waits for a first found finite $A \leqq \mathcal{G}$ such that $\psi_s(H_s)$ detaches in A as a direct summand and

$$\dim \psi_s(H_s)[p] + 1 = \dim A[p].$$

Without loss of generality, we may assume that $A = \psi_s(H_s) \oplus \mathbb{Z}_p$. Indeed, there should be an element a of the socle $\mathcal{G}[p]$ such that $\psi_s(H_s)[p]$ intersects $\langle a \rangle$ only by $\mathbf{0}$, for otherwise the socle $\mathcal{G}[p]$ would have finite dimension. Therefore, the cyclic subgroup $\langle a \rangle$ generated by a will detach in $\langle \psi_s(H_s), a \rangle$ as a direct summand with $\psi_s(H_s)$ serving as its direct complement. Then the strategy defines $\psi_{s+1}(v) = a$ and then extends the map naturally to $\langle H_s, v \rangle = H_s \oplus \langle v \rangle$. We remove v from the basis of the currently built part of $\mathbb{V}_p$, and set $H_{s+1} = H_s \oplus \langle v \rangle$. This strategy is not primitive recursive.

The copy-delay strategy

The task of the strategy is introducing more $\mathbb{Z}_p$-summands to $H_s \oplus \bigoplus_{i \leq k_s} \mathbb{Z}_p$ thus increasing k_s to a new $k_{s+1} = k_s + 1$. We keep ψ undefined on the new summands.

The copy-onto strategy

The strategy is in charge of making ψ onto. It has a witness $g \in \mathcal{G}$. When the strategy becomes active, it finds a finite $A \geqq \psi_s(H_s)$ such that $g \in A$, then extends H_s to a finite group $H_{s+1} \cong A$ using the pull-back via ψ_s. This strategy is not primitive recursive.

In the construction we let the strategies act according to their instructions, using the copy-delay strategy to pass time while waiting for either the copy-total strategy or the copy-onto strategy to become active. This ensures that $\mathcal{I}_s = \bigoplus_s H_s$ is fpr. Verifying that ψ is an onto isomorphism is straightforward.

3.2. Negative results

It is not difficult to construct an example of a computable structure that has no fully primitive recursive copy. Perhaps, the easiest such example is (ω, S, A) where s is the standard successor function on ω and A is the unary predicate coding a computable set that is not primitive recursive. We open this section with a proposition showing that such structures can be found "arbitrarily close" to fully primitive recursive ones. For this purpose we define a computation to be primitive recursive relative to a total function f if its general recursive definition can be viewed as primitive recursion with the extra symbol for f. The proof of the proposition below, although not difficult, may be of some interest to the reader as it splits into two substantially different cases.

Proposition 3.1. *Let f be total and not primitive recursive. Then there is a structure $\mathcal{A}$ with the domain ω which is primitive recursive relative to f but is not isomorphic to a fpr structure.*

Proof. Fix a computable list $\{p_n\}_{n \in \omega}$ of all primitive recursive functions such that n codes the definition of p_n by primitive recursion and composition. Let $p_{n,s}(x)$ be the result (defined or undefined) of the uniform partial computation of p_n at the stage s. Note that the function $p_{n,s}(x)$ is uniformly primitive recursive. Let $t_n(x)$ be the first number s such that $p_{n,s}(x) \downarrow$. It is easy to see that each t_n, $n \in \omega$, is primitive recursive. We say that t_n is the *time* function for p_n.

We claim that it is enough to find a set A which is primitive recursive relative to f but is not primitive recursive. Once such a set is found, we define the desired structure to be the successor structure upon ω with the unary predicate coding the set.

Case 1. There exists a primitive recursive function t such that $f(x) < t(x)$ for all $x \in \omega$. Then we can set $A = \text{graph } f = \{\langle x, y \rangle : y = f(x)\}$. Now A is not primitive recursive since $f(x) = (\mu y < t(x))[\langle x, y \rangle \in A]$.

Case 2. Case 1 does not hold. Then for every primitive recursive function t there are infinitely many $x \in \omega$ such that $t(x) \leqslant f(x)$. Define $A(x) \in \{0, 1\}$, $x \in \omega$, inductively. Suppose $A(y)$, $y < x$, is already defined. Then define

$$g(x) = \begin{cases} (\mu n \leqslant x)[p_{n,f(x)}(x) \downarrow \ \& & \text{if there is an } n \leqslant x \text{ such that} \\ \neg(\exists y < x)[A(y) \neq p_{n,f(y)}(y) \downarrow]], & p_{n,f(x)}(x) \downarrow \text{ and for no } y < x \text{ we have} \\ & A(y) \neq p_{n,f(y)}(y) \downarrow, \\ 0 & \text{otherwise,} \end{cases}$$

and

$$A(x) = \begin{cases} 1, & \text{if } p_{g(x),f(x)}(x) = 0, \\ 0 & \text{otherwise.} \end{cases}$$

Clearly, $A(x)$ and $g(x)$ are primitively recursive relative to f. To show that A is not primitive recursive we prove by induction that for every n we have a y_n such that $A(y_n) \neq p_{n,f(y_n)}(y_n) \downarrow$.

Suppose that the statement holds for every $n' < n$. For the next n we consider an $x > y_{n'}$, $n' < n$, such that $t_n(x) \leqslant f(x)$. Then the inductive hypothesis and the definition of g ensures that either there exists an $y = y_n < x$ such that $A(y) \neq p_{n,f(y)}(y)\downarrow$, or $g(x) = n$. In the last case we can set $y_n = x$ by the definition of A. Thus, $A \neq p_n$ for every $n \in \omega$, so that A is not primitive recursive. □

Our next theorem gives several negative results which disprove a few natural conjectures the reader might already have at this point. The proof of Theorem 1.5(2) for torsion-free groups relies on the existence of a computable copy of the group with a computable basis. As has recently been discovered in [24], there is a broad class of commutative algebraic structures that have a computable copy with a computable basis. These also include ordered abelian groups [20], differentially closed fields, difference closed fields, and real closed fields [24]. One might be tempted to conjecture that the existence of a computable basis always implies the existence of a fpr presentation, with a proof along the lines of Theorem 1.5(2). We will see this is not true.

Every computable graph has a primitive recursive copy (the domain does not have to be ω). Also, every computable locally finite graph clearly has a fpr presentation. The natural conjecture would be that all computable graphs have fpr copies. We show that this is not true either.

Finally, the reader may think that Theorem 1.5 can be extended to all computable abelian groups, but this is again a wrong guess.

Theorem 3.2. *In each of the following classes, there exists a computable structure that does not admit a fpr presentation:*

(1) Torsion abelian groups.
(2) Archimedean ordered abelian groups.
(3) Undirected graphs.

Proof. (1). We are building a group that will be of the form

$$A_S = \bigoplus_{p \in S} \mathbb{Z}_p,$$

for some infinite set S of primes. Note that A_S has a computable copy iff S is c.e. It thus suffices to construct an infinite c.e. set of primes S such that A_S has no fpr presentation.

We construct S by stages. The diagonalization strategy N_e will be waiting for the eth fpr structure $\mathcal{P}_e$ to either give us a non-zero element a_e of some order $m > 0$, in which case it will attempt to keep at least one prime factor of m out of S, or $\mathcal{P}_e$ will give us an element of infinite order, in which case we win automatically since A_S is a torsion group. There will also be auxiliary strategies, each controlling a primary summand of A_S, that collectively ensure that S is infinite. Each auxiliary strategy will attempt to enumerate a different prime number into S.

We now explain how the strategies interact. The auxiliary strategies working below a diagonalization strategy N_e will have followers significantly smaller than the current lower bound on the order of a_e. More specifically, if the order of a_e at stage s is greater than m, then we could use a prime number less than $\sqrt[m]{m}$ for an auxiliary strategy below N_e. This ensures that the product of all followers controlled by the auxiliary strategies below N_e will be less than the order of a_e if it is finite.

If at a later stage we see that the order of a_e is equal to $p_1 \dots p_k$, then for at least one prime p_i in this factorization, all cyclic summands in $A[s']$ are not of order p_i, and we will win N_e by permanently keeping p_i out of S.

Now the strategies can be put together by a standard finite injury priority argument. We make an interesting observation about the complexity of the priority method used here. The satisfaction of N_e depends on whether the order of a_e is finite or infinite, and would normally involve infinite injury. However the Σ^0_2 outcome is a degenerate one, as the fpr nature of $\mathcal{P}_e$ allows us to either increase the lower bound on the order of a_e, or to obtain the order of a_e at each stage of the construction, so we never need to "wait forever". In this way the lower priority auxiliary strategies only need to guess if N_e is met via a Σ^0_1 or a Π^0_1 outcome.

(2). We build a computable real $\alpha > 1$ and let G be the additive subgroup of reals $(\mathbb{R}, +)$ generated by $\{1, \alpha\}$. Clearly G has a computable copy since α is computable. Note that since the group has a finite basis, two copies of it are isomorphic iff they are computably isomorphic.

We identify 1 and α with their respective numbers in G. At stage s we will have $\alpha_s \in \left[\frac{m_s}{n_s}, \frac{m_s+1}{n_s}\right]$. The diagonalization strategy $D_{e,j}$ will want to argue that P_j is not isomorphic to G via φ_e. It will wait for $\varphi_e(1)$ and $\varphi_e(\alpha)$ to converge and see whether in P_j we have

$$\varphi_e(\alpha) \in \left[\frac{m_s}{n_s}\varphi_e(1), \frac{m_s+1}{n_s}\varphi_e(1)\right],$$

or more formally,

$$m_s\varphi_e(1) \leq n_s\varphi_e(\alpha) \leq (m_s+1)\varphi_e(1).$$

If not, then the requirement is met. Another possibility is that P_e proves it is not an ordered abelian group, and this can be detected in a c.e. way since such groups are $\forall$-axiomatizable. Otherwise, enumerate the primitive structure P_j and see whether $\varphi_e(\alpha)$ belongs to the first half or the second half of the interval $[\frac{m_s}{n_s}\varphi_e(1), \frac{m_s+1}{n_s}\varphi_e(1)]$. In the definition of α_{s+1} we do the opposite. More specifically, in the former case define a new rational approximation α_{s+1} of α to be any number in the *second half* of the interval $[\frac{m_s}{n_s}, \frac{m_s+1}{n_s}]$, and in the second case choose α_{s+1} to be a rational from the *first half* of the interval. Alternatively, $\varphi_e(\alpha)$ may approach the middle of the interval, and in this case it will eventually be in the middle third of the interval; in this case choose α_{s+1} in the first third of $[\frac{m_s}{n_s}, \frac{m_s+1}{n_s}]$. In each case we will successfully diagonalize against φ_e and P_j, and the requirement will be met.

In the construction we will also need to choose a better approximation α_{s+1} of α in the case where none of the $D_{e,j}$ for $\langle e, j\rangle < s$ need to act. The construction can then be implemented by a straightforward finite injury argument.

(3). The graph $\mathcal{G}$ which we build will have a special vertex c which we call the *coloring* vertex. Let E denote the edge set of $\mathcal{G}$. A vertex $x \neq c$ is *red* if $\{x, c\} \in E$, and otherwise x is *blue*. The requirements are

$$D_{\langle n,m\rangle} : \neg(\exists \text{ isomorphism } f : (\omega, E_n) \to \mathcal{G} \text{ where } f(m) = c),$$

where E_n is the n-th binary primitive recursive relation and $m \in \omega$.

The universe of $\mathcal{G}$, except for the coloring vertex c, splits into disjoint sets of red elements

$$R_k = \left\{r_k^1, r_k^2, \ldots, r_k^{k+3}\right\},$$

and blue elements

$$B_k = \left\{b_k^1, b_k^2, \ldots, b_k^{k+3}\right\},$$

for each $k \in \omega$. The set E contains the edges $\{r_k^i, r_k^j\}$ and $\{b_k^i, b_k^j\}$ if $i - j \equiv 1 \pmod{k+3}$, connecting each of sets R_k and B_k into cycles of the size $k+3$. The set R_k will be called *the kth red cycle*, and B_k will be called *the kth blue cycle*. Note that the graph $\mathcal{G}$ will not have any edges between vertices of the same color except for those described above.

For each $k \in \omega$ the elements of the cycles R_k and B_k will either be not connected with all vertices from R_p and B_p, $p > k$, of the other color, or they will be connected with almost all of them. The construction does not connect the cycles R_k and B_k to vertices from R_p and B_p, $p > k$, of the other color unless the diagonalization strategy $D_{\langle n,m\rangle}$ (see below), where $\langle n, m\rangle = k$, acts to connect these vertices.

The main activity of the construction is to add the cycles R_k and B_k, $k \in \omega$, one at a time, while waiting for a diagonalization strategy to become active. For each $\langle n, m\rangle$ the $D_{\langle n,m\rangle}$-strategies described below work independently from each other.

The strategy for $D_{\langle n,m\rangle}$

(1) We say that an element $z \in \omega$, $z \neq m$ is $\langle n, m\rangle$-*red* if $(x, m) \in E_n$. Otherwise z is $\langle n, m\rangle$-*blue*. Wait for a $\langle n, m\rangle$-red E_n-cycle X_k of size $k+3$ and a $\langle n, m\rangle$-blue E_n-cycle Y_k of size $k+3$ to be enumerated in (ω, E_n) for each $k \leqslant \langle n, m\rangle$. If such cycles do not exist $D_{\langle n,m\rangle}$ is satisfied. Also, we meet $D_{\langle n,m\rangle}$ if there exists an E_n-cycle of the size $k+3 \leqslant \langle n, m\rangle + 3$ which differs from X_k and Y_k.

(2) Suppose we have at most N vertices added into $\mathcal{G}$ at the current stage. Fix a finite set $Z \subseteq \omega$ of size $N+1$ such that $m \notin Z$ and

$$Z \cap \bigcup_{k \leqslant \langle n,m\rangle} (X_k \cup Y_k) = \emptyset.$$

Check whether we have an integer $z \in Z$ such that either

$$z \text{ is } \langle n, m\rangle\text{-red and } (\exists y \in Y_{\langle n,m\rangle})[(z, y) \in E_n],$$

or

$$z \text{ is } \langle n, m\rangle\text{-blue and } (\exists x \in X_{\langle n,m\rangle})[(z, x) \in E_n].$$

If no such $z \in Z$ exists, then we start to connect by edges each new vertex from R_p, $p > \langle n, m\rangle + 3$, with the elements of $B_{\langle n,m\rangle+3}$. Also we start to connect by edges each new blue vertex from B_p, $p > \langle n, m\rangle$, with the elements of $R_{\langle n,m\rangle}$. Then in $\mathcal{G}$ we will have at most N vertices outside of $\{c\} \cup \bigcup_{k \leqslant \langle n,m\rangle}(R_k \cup B_k)$ which are not connected with the elements of $R_{\langle n,m\rangle} \cup B_{\langle n,m\rangle}$. But for (ω, E_n) we have at least $N+1$ elements outside of $\{m\} \cup \bigcup_{k \leqslant \langle n,m\rangle}(X_k \cup Y_k)$ not connected with the elements $X_{\langle n,m\rangle} \cup Y_{\langle n,m\rangle}$. Thus, $D_{\langle n,m\rangle}$ is satisfied.

(3) If such $z \in Z$ exists, then we satisfy $D_{\langle n,m\rangle}$ just because we will not connect the vertices of $R_{\langle n,m\rangle}$ and $B_{\langle n,m\rangle}$ with vertices from R_p and B_p, $p > \langle n, m\rangle$.

It is easy to see that the graph $\mathcal{G}$ is computable, and the strategy ensures that it is not isomorphic to any fpr graph. □

We suspect that one could get satisfactory (positive or negative) results for other common algebraic classes, the most promising classes perhaps being real closed and differentially closed fields, but we leave it open.

3.3. Strongly fpr presentations

In this subsection we briefly discuss one possible strengthening of our main definition. We first explain the motivation. Although in the next section we will essentially show that almost all structures from Theorem 1.5 posses complicated fpr copies, the reader may find the *queuing idea* (exploited in the proof of Theorem 1.5 for delaying) somewhat unsatisfying. Indeed, we perhaps want our structure to be rapidly growing not only *globally* but also *locally*. We may remove this delaying feature in many structures by considering the natural primitive recursive analogy of 1-decidability.

Definition 3.3. A *strongly primitive recursive structure* $\mathcal{I}$ is a fpr structure which possesses a primitive recursive Skolem function.

The definition above is equivalent to saying that there exists a primitive recursive Φ such that

$$\Phi(\bar{c},\phi)=\begin{cases}-1, & \text{if } \mathcal{I}\not\models\exists x\phi(\bar{c},x),\\ y, & \text{such that } \mathcal{I}\models\phi(\bar{c},y),\end{cases}$$

where $\bar{c}\in\mathcal{I}$ and ϕ (the Gödel number of) a quantifier-free formula in the language of the structure. We note that this approach resembles the earlier notion of an *honest witness* due to Cenzer and Remmel [7].

Example 3.4. The following structures have strongly primitive recursive copies:

- The additive groups $\mathbb{Z}$ and $\mathbb{Q}$ and their direct sums.
- The countable atomless Boolean algebra.
- The order-type ω.

Clearly, there exist fpr structures that have no 1-decidable presentation, and thus have no strongly primitive recursive presentation. For instance, there exist computable linear orders in which the successivity relation is intrinsically undecidable [11], and similarly there exists a computable Boolean algebra in which the atom relation is intrinsically undecidable [19]. Now Theorem 1.5 guarantees that in each of these classes we can find fpr presentation, but no strongly p.r. presentations can exist. However, these examples are unsatisfying since they all give fpr structures that are not even 1-decidable. A rather straightforward example below separates strongly primitive recursive structures from 1-decidable fpr structures.

Proposition 3.5. *There exists a fpr* 1*-decidable equivalence structure that has no strongly primitive recursive presentation.*

Proof. For any infinite set X, let $E(X)$ denote the equivalence structure having exactly one class of size x for each $x\in X$. Note that for an infinite c.e. set X, the structure $E(X)$ has a computable, hence fpr presentation (by Theorem 1.5(1)). To make $E(X)$ 1-decidable, make the kth class have size exactly x_k, where $X=\{x_0,x_1,x_2,\ldots\}$ is some computable enumeration of X. It is easy to see that deciding an existential formula about $\bar{c}$ boils down to deciding the sizes of the classes that contain $\bar{c}$.

Thus, it remains to build an infinite c.e. set X (in fact, X will be computable) such that $E(X)$ has no strongly p.r. presentation. Suppose we have enumerated $\{x_0,\ldots,x_k\}$. Suppose we want to diagonalize against the eth potential strongly fpr structure S_e. When S_e is first processed we use the primitive recursive Skolem function in S_e to primitively recursively decide if there exists a class $[z]$ of size $>s$. If no then we win because S_e must contain at least two classes of equal sizes, and thus $E(X)\ncong S_e$. If yes then we can primitively recursively compute a witness z. In this case we say that S_e is pending with witness z. We will ensure that all future elements of X are chosen to be smaller than the current approximation to the size of $[z]$ in S_e.

At stage s of the construction we process each requirement S_e for $e<s$. If S_e is unstarted then we proceed as above, and move to the next requirement. If S_e is already pending with witness z we check if the size of $[z]$ in S_e is larger than s. If yes, the status of S_e remains pending, and we move to the next requirement. If no then the size of $[z]$ must be s. In this case we terminate the actions of stage s at S_e and initialize all lower priority requirements.

It is easy to see that if s is enumerated in X at stage s then this is compatible with the satisfaction of all requirements R_e, $e<s$. Each requirement is initialized finitely often and will be met. Finally X is infinite because only a pending requirement can block the enumeration of s into X at stage s. □

Given a class K of structures we may ask whether every fpr 1-decidable member of this class K has a strongly fpr presentation. It might also be interesting to develop fpr computable structure theory specifically for strongly fpr structures since they are perhaps closer to being genuinely "computable without delay". *We leave these questions and directions open.*

4. fpr-Categoricity

Recall that a structure is fpr-categorical if it has a unique fpr presentation up to fpr isomorphism (Section 1.4). Recall also that an isomorphism f is fpr if f and f^{-1} are both primitive recursive. We first illustrate this definition with several simple examples. Let $(p_e)_{e\in\omega}$ be a computable listing of all primitive recursive functions.

Example 4.1.

(1) The additive group $\mathbb{V}_p \cong \bigoplus_{i\in\omega} \mathbb{Z}_p$ is fpr-categorical

Proof. Indeed, suppose $\mathcal{A}$ and $\mathcal{B}$ are fpr presentations of $\mathbb{V}_p$. Note that given a tuple $\bar{a}$ in $\mathcal{A}$ we can primitively recursively choose a maximal $\mathbb{Z}_p$-independent sub-tuple in $\bar{a}$. We may assume that we always choose the lexicographically smallest independent tuple among all such independent sub-tuples of $\bar{a}$. We first explain how we can define a primitive recursive isomorphism $f : \mathcal{A} \to \mathcal{B}$, and then we explain how we make sure that f^{-1} is primitive recursive as well. Suppose $f : \bar{a}_s \to \bar{b}_s$ has already been defined, where $\bar{a}_s$ is the longest initial segment of $\mathcal{A}$ on which f has been defined. To define $f(a)$ on the next element a of $\mathcal{A}$ extending $\bar{a}_s$, first see whether a is dependent on $\bar{a}_s$. If yes, then suppose $a = \sum_j m_j a_j$, where the a_j range over $\bar{a}_s$ and m_j over $\mathbb{Z}_p$. In this case set $f(a) = \sum_j m_j f(a_j)$. If no, then we look through at most $p^{card(b_s)}$ first elements of $\mathcal{B}$ and choose the first found element b independent over $\bar{b}_s$. To make sure f^{-1} is primitive recursive as well, we choose the longest initial segment $\bar{b}'_s$ for which f^{-1} has been defined and repeat the procedure above but now with a and f replaced by b and f^{-1}. □

(2) The dense linear order $(\mathbb{Q}, <)$ without end points is *not* fpr-categorical

Proof. We produce fpr copies $\mathcal{A}$ and $\mathcal{B}$ and diagonalize against all pairs of primitive recursive $(p_i, p_j)_{i,j\in\omega}$, where p_j plays the role of a potential p_i^{-1}. We explain how to diagonalize against the first pair (f, g). Begin by growing increasing chains $a_0 < a_1 < a_2 < \cdots$ and $b_0 < b_1 < b_2 < \cdots$ in $\mathcal{A}$ and $\mathcal{B}$ respectively, and wait for $f(a_0)$ to halt. When we see $f(a_0)\downarrow = b_i$, keep growing $\mathcal{A}$ and $\mathcal{B}$ in the same way, but in $\mathcal{B}$ we add one additional point $b^* < b_0$. Now wait for g to converge on b^*. In order for $g = f^{-1}$ we must have $g(b^*) < a_0$ in $\mathcal{A}$, but there are currently no elements in $\mathcal{A}$ with this property, so we win against the pair (f, g).
For a general requirement suppose we have built $\hat{a}_0 < \hat{a}_1 < \cdots < \hat{a}_k$ in $\mathcal{A}$ and $\hat{b}_0 < \hat{b}_1 < \cdots < \hat{b}_k$ in $\mathcal{B}$. We now wish to attack (p_i, p_j). Begin as above by growing $\hat{a}_k < a_0 < a_1 < \cdots$ and $\hat{b}_k < b_0 < b_1 < \cdots$. We wait for $p_i(\hat{a}_0)\downarrow$. When we see this, we add a new point $b^* < \hat{b}_0$ and wait for $p_j(b^*)\downarrow$. Since each pair (p_i, p_j) are total functions we can finish each pair in this way before moving on to the next pair. In between satisfying each requirement, we can extend $\mathcal{A}$ to the left and make progress towards making $\mathcal{A}$ and $\mathcal{B}$ dense. □

(3) The successor structure $\mathcal{S} = (\omega, S)$, where $S(x) = x + 1$, is *not* fpr-categorical

Proof. Build two fpr copies $\mathcal{A}$ and $\mathcal{B}$ of $\mathcal{S}$ and enumerate all potential primitive recursive isomorphisms p_e. The copy $\mathcal{B}$ is the standard copy, and $\mathcal{A}$ will be used to diagonalize against primitive recursive isomorphisms. The strategy for diagonalizing p_e is the following. Pick a fresh witness $x \in \mathcal{A}_s$ which currently has no predecessor, and wait for $p_e(x)$ to converge. While waiting we grow two independent chains, one with the least element $0_{\mathcal{A}}$ and the other with the least element x. That is, introduce distinct elements $S(0_{\mathcal{A}}), S^2(0_{\mathcal{A}}), S^3(0_{\mathcal{A}}), \cdots$ and distinct elements $S(x), S^2(x), S^3(x), \cdots$. When we see $p_e(x)$ converge, declare x to be $S^n(0_{\mathcal{A}})$ for the least $n > p_e(x)$. □

Examples (1) and (2) illustrate the subtle difference between computable and primitive recursive back-and-forth methods. Interestingly (1) shows that the usual computable back-and-forth construction still works for $\mathbb{V}_p$ in the fpr setting, while (2) in contrast shows that the back-and-forth method cannot be adapted for $\mathbb{Q}$. Example (3) shows that there exists rigid computably categorical structures that are not fpr-categorical. As we will see later (Proposition 4.2), there also exist rigid infinite fpr-categorical structures, which are less straightforward to construct.

4.1. fpr-Categoricity in natural algebraic classes

We now characterize the notion of fpr-categoricity in several common algebraic classes. The proof of the theorem below exploits Theorem 1.5 and techniques from its proof. It also applies several classical results from computable structure theory. Recall that, according to our definition, a fpr-categorical structure must in particular have a fpr presentation.

Proof of Theorem 1.7. (1). Note that if an equivalence structure S has one of the claimed isomorphism types then it is fpr-categorical.

Now suppose S is fpr-categorical. First, assume the structure has infinitely many classes each having at least two elements. We fix a fpr presentation $\mathcal{A}$ of S and build $\mathcal{B}$. To diagonalize against a primitive recursive f, we iterate the following strategy. In $\mathcal{B}$ we only put classes of size 1 that we may (or may not) later grow to a larger size, and wait for some class of size at least 2 to show up in $\mathcal{A}$. Such a class must eventually show up, by our assumption. Let's say this class contains $\{a_0, a_1\}$. Next we wait for $f(a_0)$ and $f(a_1)$ to converge. When these two converge, we see that f cannot be an isomorphism (as we have only been putting classes of size one in $\mathcal{B}$). Only after then we grow the size one classes from $\mathcal{B}$ to be isomorphic to $\mathcal{A}$ keeping $f(a_0)$ and $f(a_1)$ in different classes.

In general, at step e of the construction we assume that we have defined B up to the first s many elements, and $A \upharpoonright s \cong B \upharpoonright s$. To kill off the next primitive recursive function p_{e+1} we continue putting new classes in $\mathcal{B}$ of size one, while waiting for $s+1$ many new classes of size at least 2 to show up in A, and for p_{e+1} to converge on all of these classes. Then p_{e+1} has to map at least one of these classes outside $\mathcal{A} \upharpoonright s$, and we have diagonalized against p_{e+1}. We then grow $\mathcal{B}$ to match $\mathcal{A}$ on these new elements.

Thus, the equivalence structure S has either only finitely many different classes, or almost every class is of size 1. We now claim that if S contains an infinite class C, then in fact S has only finitely many elements outside of C. This implies that S is one of the claimed isomorphism types. For a contradiction, suppose that $S - C$ is infinite. We fix a fpr presentation $\mathcal{A}$ of S and build $\mathcal{B}$. To diagonalize against a primitive recursive f, we assume that we currently have $(\mathcal{A} - C) \upharpoonright s \cong (\mathcal{B} - C) \upharpoonright s$. Now in $\mathcal{B}$ we continue adding elements to the class C and hold back from adding elements to $\mathcal{B} - C$. We wait for $s+1$ many new elements to be enumerated in $\mathcal{A} - C$, and for $s+1$ many new elements to be enumerated in C in $\mathcal{A}$, and for f to be defined on all of them. Now f has to map at least one of the new elements of $\mathcal{A} - C$ to an element of C in $\mathcal{B}$, and at least one of the new elements of C in $\mathcal{A}$ to an element of C in $\mathcal{B}$, and so we have diagonalized against f. Now grow $\mathcal{B} - C$ to catch up with $\mathcal{A} - C$. We can iterate the above procedure to diagonalize against all primitive recursive $(p_e)_{e\in\omega}$.

(2). First we prove that if a linear ordering $\mathcal{L}$ is not computably categorical then it is not fpr-categorical. This claim can be easily deduced from the proof of Theorem 1.5(2). Indeed, suppose $\mathcal{L}_0, \mathcal{L}_1$ are computable copies of the linear order $\mathcal{L}$ that are not computably isomorphic. The proof of Theorem 1.5(2) guarantees that unless $\mathcal{L} \cong \omega + \mathbb{Z} \cdot \eta + \omega^*$ or $\omega + \omega^*$ is a sub-interval of $\mathcal{L}$, there exist two fpr copies I_0 and I_1 of $\mathcal{L}$ such that I_0 is *computably* isomorphic to $\mathcal{L}_0$ and I_1 is *computably* isomorphic to $\mathcal{L}_1$. We claim that if $\mathcal{L} \cong \omega + \mathbb{Z} \cdot \eta + \omega^*$ or $\mathcal{L}$ contains a sub-interval $\omega + \omega^*$, then $\mathcal{L}$ is not fpr-categorical.

Suppose $\mathcal{L} \cong \omega + \mathbb{Z} \cdot \eta + \omega^*$. Build two fpr copies, X and Y. To diagonalize against p_e, we keep building the left-most ω-chain in X and the right-most ω^*-chain in Y and wait for p_e to converge on some $x \in X$ that is in the ω-chain but its image is in the ω^*-chain of Y. Such an element must eventually be found. We then do a few more steps towards making X and Y isomorphic to $\omega + \mathbb{Z} \cdot \eta + \omega^*$, and repeat the strategy with p_{e+1}.

Now assume $\mathcal{L}$ contains a sub-interval $\omega + \omega^*$, say $[c, d]$. We can implement essentially the same strategy as in the previous case, but now extending the ω-chain in X and the ω^*-chain in Y while we wait for p_e to converge. Note that at intermediate stages we have to also build an isomorphism from X to $\mathcal{L}$, and similarly from Y to $\mathcal{L}$. But this can be done since we *know* that $[c, d] \cong \omega + \omega^*$. We can simply keep building the respective interval in X (in Y) while we wait for another element outside $[c, d]$ to show up in $\mathcal{L}$. As soon as a new element outside $[c, d]$ appears in $\mathcal{L}$, we copy it into X and Y. We then proceed to meeting the next diagonalization requirement, then copy another element, etc. We leave the elementary details to the reader.

Now if $\mathcal{L}$ is an infinite computably categorical linear order, then it contains finitely many successivities [11]. In this case any isomorphism has to map each η-interval to the corresponding η-interval. Thus, the same argument as in Example 4.1(2) illustrates that $\mathcal{L}$ is not fpr-categorical.

(3). We assume $\mathcal{B}$ is infinite. The proof splits into several cases.

First, suppose $\mathcal{B}$ is infinite and computably categorical. Then $\mathcal{B}$ it has only finitely many atoms $e_0, \cdots, e_k$ [32,19] and at least one atomless element d. We construct two fpr copies $\mathcal{A}$ and $\mathcal{C}$ of $\mathcal{B}$ with no fpr isomorphism between them. To diagonalize against a pair of primitive recursive functions $p : \mathcal{A} \to \mathcal{C}$ and $q : \mathcal{C} \to \mathcal{A}$, fix a witness w in the dense part of $\mathcal{A}$ which is currently an atom in $\mathcal{A}_s$, i.e. we have not yet split w in $\mathcal{A}_s$. We do not yet split w in $\mathcal{A}$ and wait for $p(w)$ to converge. If $p(w) \in \mathcal{C}$ is equal to one of the finitely many (true) atoms in $\mathcal{C}$, then resume splitting w in $\mathcal{A}$. Otherwise, immediately split $p(w)$ in $\mathcal{B}$ if it already is not split, say $p(w) = c \oplus d$. Do not resume splitting w in $\mathcal{A}$ until $q(c)$ and $q(d)$ converge (but keep building $\mathcal{A}$ elsewhere). Note that $q(c) \vee q(d) \neq w$. As soon as they converge, resume splitting w in $\mathcal{A}$. The strategy can be iterated to diagonalize against all pairs of primitive recursive functions.

Now suppose $\mathcal{B}$ is not computably categorical and has an atomless element. In this case the isomorphism produced in the proof of Theorem 1.5(4) is computable. We thus can produce two fpr presentations of $\mathcal{B}$ that are not even computably isomorphic.

Finally, suppose $\mathcal{B}$ is not computably categorical and is atomic. This is equivalent to saying that $\mathcal{B}$ is atomic and infinite. Unfortunately, in this case we cannot use Theorem 1.5(4) as its proof does not produce a computable isomorphism. Nonetheless, we will combine the *basic copying strategy* and the *isomorphism-builder strategies* from the proof of Theorem 1.5(4) with a simplified version of the above diagonalization strategy, as follows.

We construct two fpr copies, $\mathcal{A}$ and $\mathcal{C}$, of $\mathcal{B}$. The fact that $\mathcal{B}$ is fpr but not merely computable will not be helpful. Both $\mathcal{A}$ and $\mathcal{C}$ will be copying $\mathcal{B}$ (via almost-isomorphisms $\phi_{\mathcal{A}}$ and $\phi_{\mathcal{C}}$) simultaneously diagonalizing against all potential primitive recursive isomorphisms $p_e : \mathcal{A} \to \mathcal{C}$.

The diagonalization strategy

Wait for more new atoms $d_1, d_2, \ldots$ to be introduced into $\mathcal{A}$ and more new atoms $e_1, e_2, \ldots$ to be put into $\mathcal{C}$ by the basic copying strategies (to be discussed) working with $\mathcal{A}$ and $\mathcal{C}$, respectively. Wait for more of the $p_e(d_1), p_e(d_2) \ldots$ to converge. If $p_e(d_1), \ldots, p_e(d_k)$ are not distinct atoms in $\mathcal{C}$ for some k, then stop. Otherwise, there must be a k (large enough) such that $p(d_k) = e_i$ for some new atom enumerated by into $\mathcal{C}$. Once such an e_i is found, split $p(d_k) = e_i$ into two disjoint elements x_i and y_i and stop.

The isomorphism-builder strategies

For both $\mathcal{A}$ and $\mathcal{C}$, the strategy is the same as in the proof of Theorem 1.5(4). We also adopt the same priority and initialization as in the proof of Theorem 1.5(4).

The basic copying strategy

This is the same as in the proof of Theorem 1.5(4) for $\mathcal{A}$, with the only extra promise that the atom d_k used for the diagonalization purpose will be put into the queue (and thus will be kept an atom). The strategy working in $\mathcal{C}$ will also need a slight modification. Recall at most one of the new atoms e_i produced by the strategy in $\mathcal{C}$ may be split into two new atoms, x_i and y_i, due to an action of some diagonalization strategy. When the copying strategy in $\mathcal{C}$ forms a queue, it will put these new elements x_i, y_i into the queue instead of e_i.

The basic copying strategy

Alternate between building $\mathcal{A}$ and $\mathcal{C}$, as follows. In $\mathcal{A}$ we let the basic copying strategy act between splitting stages, but before the strategy stops producing more atoms it also waits for one more diagonalization strategy to finish its action. Similarly in $\mathcal{C}$, before the basic copying strategy waits for one more diagonalization strategy to finish its action. We then also let the isomorphism-builder strategies act in each $\mathcal{C}$ and $\mathcal{A}$. At every splitting stage we let the isomorphism-builder strategies re-define $\phi_{\mathcal{A}}$ and $\phi_{\mathcal{C}}$ if necessary. If an isomorphism-builder strategy I whose queue follows x is forced to change its witness, then we first initialize all weaker priority strategies working below x and only then we let the strategy I act.

Verification

We argue, that every diagonalization strategy successfully diagonalizes against its p_e. Indeed, when it first becomes active there will be at most finitely many elements in $\mathcal{C}$ previously produced by other strategies. Since different atoms must go to different atoms, eventually either p_e proves that it is not an isomorphism or the strategy finds a non-restrained element to diagonalize. The diagonalization will be successful since we will put the atom d_i in $\mathcal{A}$ into a queue, and thus it will never be split again.

Checking that $\mathcal{A} \cong \mathcal{C} \cong \mathcal{B}$ is literally the same as in Theorem 1.5(4). Indeed, the only extra effect the diagonalization strategies have in $\mathcal{A}$ is just perhaps making the queues of the extra atoms longer (due to a longer delay). But in $\mathcal{C}$ the effect is essentially the same, but now the queues might be longer because of the longer wait and also because at most one extra atom might be split into two extra atoms. Finally, both $\mathcal{A}$ and $\mathcal{C}$ are clearly fpr.

(4). Note that if a p-group $\mathcal{A}$ is not computably categorical, then its socle $\mathcal{A}[p]$ is infinite. Indeed, otherwise $\mathcal{A}$ is isomorphic to a finite sum of a finite group and the groups Z_{p^∞} by the proof of Theorem 1.5(5), and all such groups are computably categorical. But if the socle is infinite then the proof of Theorem 1.5(5) gives a computable isomorphism from any computable copy of $\mathcal{A}$ onto a fpr presentation of the group. In this case we can produce two fpr copies of $\mathcal{A}$ that are not even computably isomorphic.

Now suppose that $\mathcal{A}$ is computably categorical. Then $\mathcal{A}$ is of the form $F \oplus S$, where $F[p]$ is finite (see the description of such groups in the proof of Theorem 1.5(5)) and $S \cong \bigoplus_{k \in I} \mathbb{Z}_{p^m}$ for some fixed $m \in \omega \cup \{\infty\}$ [3,13] and some $I \subseteq \omega$. We split this situation into several subcases, according to the isomorphism type of $\mathcal{A}$.

If $\mathcal{A}$ has a divisible component, then pick a witness w in the divisible component of the first fpr presentation and do not declare its order until we see $p_e(w)\downarrow$. Then make sure that the order of $p_e(w)$ (in the second component), is different from the order of w (in the first component). Note we can control the former.

We conclude that $\mathcal{A}$ must be of the form $F \oplus S$, where F is finite and $S \cong \bigoplus_{k \in I} \mathbb{Z}_{p^m}$ for some fixed $m \in \omega$. If I is infinite then we claim $m = 1$. Indeed, if $m > 1$ then we can diagonalize against p_e similarly to how it was done in the respective case of the proof for equivalence structures (see (1) above). Indeed, we can extend the second copy by growing only its socle, but at the same time extend the first copy naturally. We wait until for some w with $order(w) > p$ we have $Order(p_e(w)) \leq p$.

(5). Suppose a fpr torsion-free abelian group G has at least one non-zero element. As we discussed in the proof of Theorem 1.5(3), without loss of generality we may assume that G has a computable basis. Let H be the copy of G produced in the proof of Theorem 1.5(3). By construction, H also has a computable (in fact, primitive recursive) basis. Recall that having a computable basis is equivalent to having a linear independence algorithm [34]. This means that, given two element h, h' we can decide whether $\{h, h'\}$ is a linearly independent set. (In fact, in H this procedure is primitive recursive, but it has no use for us.)

We will produce another fpr presentation U of G and will diagonalize against all potential primitive recursive isomorphisms $p_e : U \to H$, $e \in \omega$. We will also build a computable surjective isomorphism $\phi : U \to H$.

Since we assumed that G has at least one non-zero element, the basis of H has at least one element h. We initially start building U by introducing a non-zero $u \in U$ and declaring $\phi(u) = h$. We will be extending both U and ϕ naturally, thus copying H into U via ϕ. To diagonalize against p_e, we introduce a new element w that we keep outside the domain of ϕ. We wait for p_e to converge on w and h. Meanwhile, we make progress in building U as a direct sum of $\operatorname{dom}\phi$ and the (partial) cyclic group generated by w:

$$U_s = \operatorname{dom}\phi_s \oplus \langle w\rangle_s,$$

where $\langle w\rangle_s = \{-sw, \dots, -w, 0, w, \dots, sw\}$. As soon as $p_e(h)$ and $p_e(w)$ converge, we decide whether $\{p_e(h), p_e(w)\}$ is a linearly independent set in H. We keep w outside $\operatorname{dom}\phi$ until the decision is made. Once the decision is made and we either see $\{p_e(h), p_e(w)\}$ is independent or see a reduced linear combination $kp_e(h) = np_e(w)$ (no matter what the outcome is), declare $w = m \cdot h$, where m larger than any integer mentioned so far in the construction. Then extend ϕ naturally to $\langle w\rangle_s$ by setting in particular $\phi(w) = m\phi(h)$. This definition is clearly consistent with the part of the atomic diagram listed in U_s by far, and with making ϕ an isomorphism. This action is also consistent with making U_s fpr. We need to argue that in both cases we have successfully diagonalized against p_e.

If $\{p_e(h), p_e(w)\}$ is independent, then setting $w = m \cdot h$ will ensure p_e is not an isomorphism, as any isomorphism must preserve linear independence. Now suppose $kp_e(h) = np_e(w)$ and p_e is an isomorphism. But $w = mh$ and $kh = nw$ together imply $kh = nw = nmh$, which is impossible since m is too large. Thus, in both cases we will successfully diagonalize. □

We suspect that, along the lines of the proof of (5) above, most "natural" algebraic classes of structures having characteristic 0 (in some general sense) will have only trivial fpr-categorical fpr copies. For example, we conjecture that all known classes that have the Mal'cev Property [24] have this feature.

4.2. An infinite rigid fpr-categorical structure

Recall that in both Example 4.1 and Theorem 1.7 we have seen only fpr-categorical structures that are not rigid. Indeed, all examples we have seen so far were, in some sense, far form being rigid. The reader may have started to suspect that all fpr-categorical must be like that. Nonetheless, we show that infinite rigid structures can be fpr-categorical. Understanding the proof of the proposition below should help the reader to understand the much more involved proof of Theorem 1.8 where the constructed structure will also be rigid.

Proposition 4.2. *There exists an infinite rigid fpr-categorical structure.*

Proof. The functional signature of the structure $\mathcal{A}$ consists of a constant o (which we call the *root*) and two unary functions s and c. The universe of the structure is a union of c-cycles of finite length. The s-function maps each element of any cycle to a fixed element of another cycle. The values of the s-function together with o form the *ω-chain* $o, s(o), s(s(o)), \dots, s^n(o), s^{n+1}(o), \dots$.

For a function $f : \omega \to \omega \setminus \{0\}$ define a structure $\mathcal{A}_f$ so that the element $s^n(o)$ is located in a c-cycle of size $f(n)$ for each n. The structure $\mathcal{A}_f$ has a fpr presentation if and only if the graph $\{\langle x, y\rangle : y = f(x)\}$ is a primitive recursive set.

Given an element x of the structure, we say that x has coordinates (n, m) if x is the mth element of its cycle, and x is located in the cycle attached to $s^n(o)$. Given any fpr presentation $\mathcal{B}$ of $\mathcal{A}_f$, define $\chi_{\mathcal{B}}$ to be the function which maps each element $x \in \mathcal{B}$ to the pair (n, m) where x has coordinates (n, m). Observe that $\chi_{\mathcal{B}}^{-1}$ is always primitive recursive, but $\chi_{\mathcal{B}}$ is generally not primitive recursive. (For instance, by Example 4.1, already (ω, s) is not fpr-categorical, so given x we cannot hope to quickly compute n.)

We will identify $\mathcal{A}_f$ with its *canonical* fpr presentation (ω, o, s, c) where $\chi_{\mathcal{A}_f}$ and $\chi_{\mathcal{A}_f}^{-1}$ are both primitive recursive. Then for every fpr presentation $\mathcal{B}$ of $\mathcal{A}_f$, the function $\chi_{\mathcal{B}}{}^{-1} \circ \chi_{\mathcal{A}_f}$ is a primitive recursive isomorphism from $\mathcal{A}_f$ onto $\mathcal{B}$.

Since the structure is rigid it remains to construct a fpr structure $\mathcal{A}_f$ for some primitive recursive f satisfying the requirements

$$P_n : \mathcal{B}_n \cong \mathcal{A}_f \Longrightarrow \chi_{\mathcal{B}_n} \text{ is primitive recursive,}$$

where $\mathcal{B}_n = (\omega, o_n, s_n, c_n)$ is the n-th fpr structure in the language of $\mathcal{A}_f$. We think of the functions s_n and c_n as being partial computable functions with corresponding primitive recursive time functions t_n, i.e. $s_n(x)[t_n(x)] \downarrow$ and $c_n(x)[t_n(x)] \downarrow$ for every n and x. (Although each t_n is primitive recursive, the sequence $(t_n)_{n\in\omega}$ is not uniformly primitive recursive.)

The trick to satisfying P_n is to define the function f such that for every n and m we have

$$f(\langle n, m\rangle) \in \{2n + 1, 2n + 2\},$$

where $\langle n, m\rangle = 2^n(2m+1) - 1$. We adopt this pairing function $\langle\cdot,\cdot\rangle$ since for every n the sequence $\langle n, m\rangle$ forms an arithmetic progression, and hence in every interval of size 2^{n+1} we will have a number of the form $\langle n, m\rangle$. The choice between $2n + 1$ and $2n + 2$ will depend on the enumeration of the fpr structure $\mathcal{B}_n$.

The requirements P_n work independently for each n, and there are no interactions amongst the requirements. Fix n. We now describe informally how to define $f(\langle n, m\rangle)$ for $m \in \omega$.

We begin by considering $x = 0 \in \mathcal{B}_n$ and applying s_n to x at most 2^{n+1} times to find some y_0 associated with a c_n-cycle of length $2n+1$. If y_0 is found then compute $y_1 = s_n^{2^{n+1}}(y_0)$ and see if y_1 generates a c_n-cycle of length $2n+1$. Suppose this takes u_0 many steps. While computing these values we cannot delay the definition of $f(x)$, so we simply set $f(\langle n, m\rangle) = 2n+1$ for all $m \leq u_0$. Notice that if u_0 is not defined then we will end up defining $f(\langle n, m\rangle) = 2n+1$ for all $m \in \omega$, and in that case $\mathcal{B}_n \not\cong \mathcal{A}_f$. Otherwise u_0 is eventually found, and we will ensure that for all $m > u_0$ we never have a pair of adjacent $2n+1$-cycles, i.e. we will not have $f(\langle n, m\rangle) = f(\langle n, m+1\rangle) = 2n+1$ for any $m > u_0$.

Now we will proceed to define $f(\langle n, m\rangle)$ for $m > u_0$. We now assume $x = 1$ and apply s_n at most 2^{n+2} times to x to find some y_0 which generates a c_n-cycle of the length $2n+1$. If y_0 is found then compute $y_1 = s_n^{2^{n+1}}(y_0)$ and $y_2 = s_n^{2^{n+1}}(y_1)$ and check that y_1 or y_2 generates a cycle of length $2n+1$. Suppose this process takes $u_1 > u_0$ many steps. While waiting we define $f(\langle n, m\rangle)$ to alternate between a cycle of length $2n+1$ and a cycle of length $2n+2$ for $u_0 < m \leq u_1$. Again if u_1 does not exist we end up defining $f(\langle n, m\rangle)$ to eventually alternating between cycles of length $2n+1$ and $2n+2$, and in this case demonstrate that $\mathcal{B}_n \not\cong \mathcal{A}_f$. Otherwise u_1 is eventually found, and we will ensure that for all $m > u_1$ we never have the pattern $2n+1, 2n+2, 2n+1$.

Now we set $x = 2$ and search for y_0 with cycle length $2n+1$, and y_1, y_2, y_3 with cycle lengths $2n+2, 2n+2$ and $2n+1$ respectively. Suppose this takes u_3 steps, we define $f(\langle n, m\rangle)$ to repeat the pattern $2n+1, 2n+2, 2n+2$ for $u_2 < m \leq u_3$. Repeat this for all x, each time increasing the number of successive $2n+2$ in each pattern.

If u_x is not defined for some x, then it is easy to check that $\mathcal{B}_n \not\cong \mathcal{A}_f$. Otherwise if u_x is defined for all x, then it is easy to see that u_x is bounded by the composition of t_n with primitive recursive functions. Thus given $x \in \mathcal{B}_n$ we claim that $\chi_{\mathcal{B}_n}(x) = (n, m)$ for some $n \leq u_x$. This is because the pattern $2n+1, \underbrace{2n+2, \cdots, 2n+2}_{x \text{ times}}, 2n+1$ cannot be found after u_x. So to compute $\chi_{\mathcal{B}_n}(x) = (n, m)$ we simply generate the structure $\mathcal{B}_n$, starting from the root o, the elements $s_n(o), s_n^2(o), \cdots, s_n^{u_x}(o)$ and all the attached cycles, until we find the element x. This process is bounded by u_x and f.

(More formal details will be provided in the stronger Theorem 1.8.) □

The infinite rigid fpr-categorical structure in the proposition above was in a finite functional language. It is worth noting that a fpr-categorical structure in a relational language cannot be rigid.

Proposition 4.3. *An infinite fpr-categorical relational structure cannot be rigid.*

Proof. Let p be a primitive recursive permutation on ω such that p^{-1} is not primitive recursive. Then for a fpr relational structure $\mathcal{A}$ the structure $\mathcal{B} = p^{-1}(\mathcal{A})$ is again fpr. If $\mathcal{A}$ is rigid then the unique isomorphism from $\mathcal{A}$ onto $\mathcal{B}$ is p^{-1} which is not primitive recursive. □

4.3. A fpr-categorical structure that is not computably categorical

Recall that, by definition, a fpr-categorical structure must be fpr to begin with. This section is completely devoted to a proof of the rather counter-intuitive Theorem 1.8 which says that there exists a fpr-categorical structure which is not computably categorical. (Furthermore, the language of the structure is finite and contains only four unary functional symbols.)

Proof of Theorem 1.8. We will heavily recycle the key ideas and notation of Proposition 4.2. Therefore we suggest that the reader first familiarizes himself with the proof of the elementary Proposition 4.2.

Notation

We have four unary functions in the structure $\mathcal{A}$: c, s, p and r. The structure will have the following properties.

(1) Instead of a single distinguished root (as in Proposition 4.2), we will now have an infinite set of components, each with its own root. The function r is a projection of $\mathcal{A}$ onto the set of roots. That is, an element x of $\mathcal{A}$ is called *root* iff $r(x) = x$ iff $x \in \operatorname{rng} r$.

(2) The *component* of a root x is the set C_x of all y such that $r(y) = x$. The component will resemble the structure in the proof of Proposition 4.2, but for every root x the component will be *finite*. The component with root x consists of the following pairwise distinct elements

$$\left\{c^m(s^n(x)) : n \leqslant \ell_x \,\&\, m < f_x(n)\right\},$$

where ℓ_x is the *length* of the component, and f_x is a numeric function from $\omega \upharpoonright \ell_x + 1$ to $\omega \setminus \{0\}$. The numbers (n, m) are called *coordinates* of the element $c^m(s^n(x))$ of the component C_x. The *chain* of the root x is the set

$$H_x = \{s^n(x) : n \leqslant \ell_x\} \subseteq C_x.$$

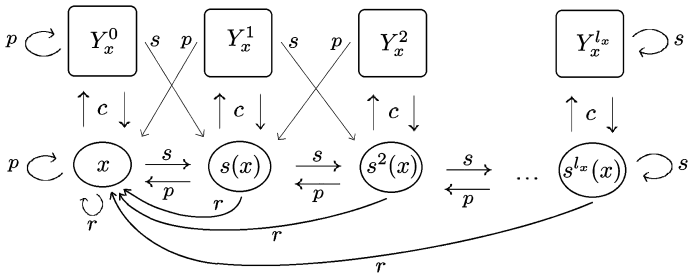

Fig. 1. A normal component with root x.

(3) The function c generates *cycles*

$$Y_x^n = \{c^m(s^n(x)) : m < f_x(n)\}$$

inside each component C_x. Y_x^n is the cycle attached to $s^n(x)$ in the chain. To close the cycles we declare

$$c^{f_x(n)}(s^n(x)) = s^n(x)$$

for every $n < \ell_x$.

(4) For the function s we have

$$s(c^m(s^n(x))) = s^{n+1}(x)$$

for $n < \ell_x$, and

$$s(c^m(s^{\ell_x}(x))) = c^m(s^{\ell_x}(x)).$$

(5) For the function p we have

$$p(c^m(s^n(x))) = s^{n-1}(x)$$

for $n > 0$.

(6) For each root x we have either

$$p(c^m(x)) = c^m(x)$$

for every $m < f_x(0)$, or

$$p(c^m(x)) = y_x$$

for every $m < f_x(0)$, where $y_x \neq x$ is some fixed root such that $p(y_x) = y_x$. In the former case we call the root x and its component *normal*. In the latter case we call the root x and its component *special* and say that the special component is *associated* with the (normal) root y_x. Every normal root has at most one special root associated with it. A normal component with root x is illustrated in Fig. 1.

(7) The value $f_x(0)$ is called the *label* of the root x. This is the length of the cycle Y_x^0 attached to the root x. The labels of two different roots can coincide only if both roots are normal and are associated with special roots of different labels. Therefore, the structure is *rigid*.

(8) For each root x we also have

$$2^i(2j+1) \leqslant \ell_x \Longrightarrow f_x(2^i(2j+1)) \in \{2i+1, 2i+2\}$$

for every i, j. This is to force the primitive recursiveness of $\chi_{\mathcal{B}}$ for each fpr presentation $\mathcal{B}$ of $\mathcal{A}$ (as in Proposition 4.2).

We build the fpr structure $\mathcal{A}$ (with uniformly primitive recursive functions $f_x(m)$) adding at stage t a new element and naming it by number $t \in \omega$ (recall the domain is ω). We also build a *computable* structure $\mathcal{B} \cong \mathcal{A}$.

Since $\mathcal{A}$ is rigid, it suffices to satisfy the requirements:

$$P_i: \quad \mathcal{B}_i \cong \mathcal{A} \Longrightarrow \text{ there exist primitive recursive isomorphisms}$$
$$h_i : \mathcal{B}_i \to \mathcal{A} \text{ and } \hat{h}_i : \mathcal{A} \to \mathcal{B}_i,$$

where $\{\mathcal{B}_i\}$ ranges over all primitive recursive structures in the language of $\mathcal{A}$, and

$$D_e: \quad \varphi_e : \mathcal{B} \to \mathcal{A} \text{ is not an isomorphism,}$$

where φ_e is the eth partial recursive function. We order the requirements according to their priority: $P_0 < D_0 < P_1 < D_1 < \cdots$.

4.3.1. Informal description of the strategies

At each stage of the construction there will be exactly one component C_x of the structure $\mathcal{A}$ which is not closed under the function s, that is, $s^n(x) \neq s^{n-1}(x)$ for the largest n where $s^n(x)$ is defined. Such a component is called *active*. All other existing components in $\mathcal{A}$ are closed. In fact we ensure p, r and c are defined on every element currently in the structure $\mathcal{A}$, and s is defined on every element except for $s^n(x)$ and the elements of Y_x^n.

To ensure that $\mathcal{A}$ is primitive recursive, at each stage s of the construction we will take one of the following actions:

- Continue growing C_x, which means defining $s^{n+1}(x)$ to be the next element in the domain of $\mathcal{A}$, and grow primitively recursively the entire cycle Y_x^{n+1} attached to $s^{n+1}(x)$, or
- close off C_x and introduce a new component C_y with a new element y as the root, and grow primitively recursively the entire cycle Y_y^0 attached to y.

We define all functions p, r, c on the new elements right away. In order to be able to do this, we will of course need to declare right away whether the new component C_y is normal or special (and in the latter case, also decide the normal component C_y is attached to).

Each requirement P_e will start off being in the *defining* state, where it will follow a certain procedure to decide the length of cycles in the currently active component. It may (or may not) later transit to being in the *diagonalizing* state. In the latter state there will be a $\mathcal{B}_e$-component C where P_e is making progress in diagonalizing against $\mathcal{B}_e$ using component C, as follows. In this case P_e is waiting for the label of C to converge, i.e. the cycle attached to the root of C to close (more detail later below).

A requirement D_e will be in one of the following states:

(S1) unstarted,
(S2) searching for labels,
(S3) pending,
(S4) diagonalizing,
(S5) finished.

At every stage s of the construction there will be exactly one D_e which is *given control*. If control is given to D_e then it will either be in state $S2$ where it is searching for suitable labels for later use, or in state $S4$ where it will embark on the final part of its strategy to "kill" φ_e by growing different components in $\mathcal{A}$ and $\mathcal{B}$. The structure $\mathcal{B}$, unless otherwise stated, will copy $\mathcal{A}$. A D_e-strategy in control and in state $S4$ may request to *freeze* $\mathcal{B}$. In this case we stop growing $\mathcal{B}$ until D_e is initialized or transits to $S5$ in which case we will resume growing $\mathcal{B}$. We will ensure that each D_e spends a finite amount of time in state $S4$ and so $\mathcal{B}$ is never frozen forever, hence, $\mathcal{B} \cong \mathcal{A}$.

Informal description of P_e

We now describe the action of P_e in isolation. The presentation in the formal construction will slightly differ what we present here (this will be done in order for the different strategies to fit together). Nevertheless, this discussion describes the basic working module for each requirement in isolation fairly accurate. Suppose that P_e is in the defining state, and that the construction has decided to enumerate a new component C_y with label a. Then P_e will wait for the previously active cycle $C_x^{\mathcal{B}_e}$ to close in $\mathcal{B}_e$, and for a new component to show up in $\mathcal{B}_e$. This wait will terminate because we will eventually see that $C_x^{\mathcal{B}_e}$ will either close itself isomorphically, or grow with a chain of size larger than the corresponding chain in $\mathcal{A}$. In the latter case we know that $\mathcal{A} \ncong \mathcal{B}_e$ and we can abandon the wait. In the former case we will check the label of the new component in $\mathcal{B}_e$, and if the label of this new component grows longer than a we will declare that P_e now enters the diagonalizing state. (We will discuss the diagonalizing state later.)

Now assume that P_e is in the defining state, and the construction has decided to keep growing the current component C_x. In this case P_e must then advise the construction about the length of the next cycle in C_x. We use a modified form of the strategy in Proposition 4.2 to do this. In Proposition 4.2 we defined u_z to be the number of steps needed to search for a pattern of $2n+1,\ \underbrace{2n+2, \cdots, 2n+2}_{z \text{ times}},\ 2n+1$ by repeatedly applying s to z. This will not work well here because each component has a finite chain. So if $z \in C_x^{\mathcal{B}_e}$ then it might be that after repeatedly applying s to z to search for the pattern above, we find some number n such that $s^{n+1}(z) = s^n(z)$. We cannot force the component C_x to close at u_x because the component is required to be kept open due to other (D_e) strategies; in general the length of each component C_x will depend on the structures $\mathcal{B}_i$ for various i. If this is the case then computing u_z tells us nothing about the coordinates of z in C_x.

The most straightforward way around this problem is to repeatedly apply p (instead of s) to z to obtain u_z. If we prematurely reach the root $x^{\mathcal{B}_e}$ of $C_x^{\mathcal{B}_e}$ before finding the pattern $2n+1,\ \underbrace{2n+2, \cdots, 2n+2}_{\leq z \text{ times}},\ 2n+1$, then we know the exact n-coordinate of z within the component $C_x^{\mathcal{B}_e}$. Otherwise we find the pattern (where $2n+2$ is repeated $\leq z$ times) in

u_z many steps, and the construction will then switch to the pattern $2n+1,\ \underbrace{2n+2,\cdots,2n+2}_{z+1 \text{ times}}$ for $m > u_z$. This allows us to argue that the coordinates of each z are primitive recursive.

Suppose now that P_e is in the diagonalizing state, waiting for the labeling loop of some $\mathcal{B}_e$-component to close. If the chain grows to infinity, then $\mathcal{B}_e \not\cong \mathcal{A}$ as $\mathcal{A}$ does not contain an infinite c-chain. Otherwise, if the label closes at some stage, then restrain this label a from the structure $\mathcal{A}$, i.e. no new component in the future will have label a. Hence $\mathcal{B}_e \not\cong \mathcal{A}$.

Now P_e is satisfied because if $\mathcal{A} \cong \mathcal{B}_e$ then P_e is always in the defining state. When new components are introduced in $\mathcal{A}$ we always wait for the previous $\mathcal{B}_e$-component to close and for the new (necessarily isomorphic) $\mathcal{B}_e$-component to show up. Hence, the set of $\mathcal{A}$-roots can be matched primitive recursively with the set of roots in $\mathcal{B}_e$. Furthermore the non-root elements can be matched because we can primitive recursively compute the coordinates of each element in $\mathcal{B}_e$.

Informal description of D_e

We now describe the strategy for D_e. It starts off in state $S1$ until it is given control by the construction. It is then upgraded to state $S2$ where it will pick a fresh number a. The construction then continues growing the current component C_x and waits for all P_i, $i \le e$ which are in their diagonalizing state to show that their respective diagonalizing component has label larger than $a+2$. If any of these P_i sees their diagonalizing label converge, we initialize D_e. This wait is finite, as each c_i, $i \le e$ is total, although we cannot delay the definition of $\mathcal{A}$, so we have to keep the current component C_x open (i.e., growing) while we wait.

Once D_e sees that all higher priority diagonalizing P-requirements have labels larger than $a+2$, it closes off the currently active component. Then the strategy introduces a new normal component $C(D_e)$ with the same label a. The construction now waits for all P_i, $i \le e$ which are in defining state to close off the old component and catch up with a new component with label a (as in the description of P's strategy above). At any stage if something goes wrong for P_i we put P_i into the diagonalizing state and initialize D_e. As mentioned above (in P's strategy) this wait is finite and if we do not initialize D_e we will get approval from all P_i, $i \le e$. At this point we will let D_e enter the pending state $S3$, and D_e relinquishes control to another (lower priority) D-strategy.

In state $S3$ we will check, at the beginning of every stage of the construction, whether control needs to be handed back to D_e. This will be the case if we see that the corresponding component $C(D_e)^{\mathcal{B}}$ in $\mathcal{B}$ with label a is mapped under φ_e to the isomorphic component $C(D_e)$ in $\mathcal{A}$. (Note that $C(D_e)$ may no longer be the active component at this point, as other D requirements might have introduced new components in the meantime.)

If φ_e converges we return control to D_e, which will move into the diagonalizing state $S4$. It will now request for $\mathcal{B}$ to be frozen, where we will not grow $\mathcal{B}$ until D_e is initialized or D_e is finished (i.e., in state $S5$). In this state $S4$ we will close off the currently active component, and introduce a new special component $C_0(D_e)$ with label $a+1$ attached to $C(D_e)$. Again wait for all P_i, $i \le e$ in the defining state to respond (by closing off their old components and growing a new one with label $a+1$). At any point if some P_i enters the diagonalizing state we initialize D_e and unfreeze $\mathcal{B}$. Otherwise all P_i, $i \le e$ have now a special component $C_0(D_e)^{\mathcal{B}_i}$ with label $a+1$ attached to $C(D_e)^{\mathcal{B}_i}$. At this point the construction will close off the special component $C_0(D_e)$ and simultaneously grow two new components $C^*(D_e)$ and $C_0^*(D_e)$. $C^*(D_e)$ will be a normal component with label a and identical with $C(D_e)$, while $C_0^*(D_e)$ will be a special component attached to $C^*(D_e)$ with label $a+2$. We will let $C_0^*(D_e)$ be the currently active component of the construction. (Note that $C(D_e)$ is already completely determined by now and so we can grow the entire component $C^*(D_e)$ primitively recursively.)

Again we wait for all P_i, $i \le e$ to catch up with isomorphic $C^*(D_e)^{\mathcal{B}_i}$ and $C_0^*(D_e)^{\mathcal{B}_i}$. We claim that this wait is again finite. Indeed, suppose the first new component we see in P_i has label $a+2$. Then, applying p_i to the root of $C_0^*(D_e)^{\mathcal{B}_i}$, we can force P_i to reveal (what must necessarily be) the root of $C^*(D_e)^{\mathcal{B}_i}$. Otherwise if the first new component we see in P_i has label a then we wait for P_i to close off $C^*(D_e)^{\mathcal{B}_i}$ and the next new component must be $C_0^*(D_e)^{\mathcal{B}_i}$. If something goes wrong at any time we move P_i to diagonalizing state and initialize D_e and unfreeze $\mathcal{B}$.

Finally, if all P_i have caught up, we will move D_e to the finished state $S5$. We unfreeze $\mathcal{B}$ and grow $\mathcal{B}$ differently: To the component $C(D_e)^{\mathcal{B}}$ we attach a special component $C_0^*(D_e)^{\mathcal{B}}$ with label $a+2$, and we grow a new normal component $C^*(D_e)^{\mathcal{B}}$ identical with $C(D_e)^{\mathcal{B}}$ but attach the special component $C_0(D_e)^{\mathcal{B}}$ with label $a+1$ to $C(D_e)^{\mathcal{B}}$. Since φ_e maps $C(D_e)^{\mathcal{B}}$ to $C(D_e)$, it cannot be an isomorphism. Hence D_e is satisfied. Note that D_e spends only a finite amount of time in $S4$ before it is initialized or finished, and so we never freeze $\mathcal{B}$ forever.

4.3.2. The formal construction

We make a technical comment. It will be convenient to view each primitive recursive function as a partial computable function φ_e where $\varphi_e(x)\downarrow$ in $p(x)$ many steps for some primitive recursive p. The structure $\mathcal{B}_e[s]$ *evaluated at stage* s refers to the finite substructure of $\mathcal{B}_e[s]$ with all $\mathcal{B}_e$ functions evaluated up to s steps. Our construction then defines partial recursive functions p, c, s, r for $\mathcal{A}$ and ensures that $p(x), c(x), s(x)$ and $r(x)$ all converge within x many steps (of the construction). Hence $\mathcal{A}$ is primitive recursive. At stage s of the construction we are only allowed to look at $\mathcal{B}_i[s]$ for $i < s$. This allows us to apply the Church–Turing thesis for primitive recursion.

Suppose we are at stage s of the construction. Let D_{e_s} be the requirement in control. The construction consists of two *phases*.

Phase 1, checking if some requirement needs attention

We now define what it means (for a requirement) to require attention. For requirement P_e this means that

- if P_e is in the defining state, then $\mathcal{B}_e[s]$ is not a substructure of $\mathcal{A}[s]$, and
- if P_e is in the diagonalizing state with component $C^{\mathcal{B}_e}$, then the label of $C^{\mathcal{B}_e}$ has converged.

For a requirement D_e, this means that

- if D_e is in state $S2$ waiting on label a, then for every P of higher priority in the diagonalizing state with component $C^{\mathcal{B}}$, the label of $C^{\mathcal{B}}$ has grown larger than $a+2$. If $C(D_e)\downarrow$ then all higher priority P in the defining state have found a matching component $C(D_e)^{\mathcal{B}}$.
- If D_e is in pending state $S3$, then φ_e has mapped some $\mathcal{B}$-component $C(D_e)^{\mathcal{B}}$ isomorphically onto $C(D_e)$ in $\mathcal{A}$.
- If D_e is in diagonalizing state $S4$ and $C^*(D_e)$ is not yet defined, then all higher priority P in the defining state have found matching components $C(D_e)^{\mathcal{B}}$ and $C_0(D_e)^{\mathcal{B}}$. Otherwise if $C^*(D_e)\downarrow$, then all higher priority P in the defining state have found matching components $C(D_e)^{\mathcal{B}}$, $C_0(D_e)^{\mathcal{B}}$, $C^*(D_e)^{\mathcal{B}}$, and $C_0^*(D_e)^{\mathcal{B}}$.

During the construction when we *pass control to the next available* D we mean that we pick the highest priority D which is currently unstarted. Declare D to be now in state $S2$ and waiting on a fresh label a. Declare that control is now given to D.

We check if there is some requirement P_i or D_i for $i \le e_s$ which requires attention at stage s. Pick the highest priority requirement Q which requires attention at s. Initialize all requirements of lower priority than Q (if $\mathcal{B}$ is frozen by D_{e_s} and D_{e_s} is initialized, unfreeze $\mathcal{B}$ and grow $\mathcal{B}$ to catch up with $\mathcal{A}[s]$). We then take the following steps to give Q attention.

$Q = P_e$ **in the defining state:** Declare P_e to be now in the diagonalizing state, with the currently active (unclosed) $\mathcal{B}_e$-component as the diagonalizing component. Pass control to the next available D.

$Q = P_e$ **in the diagonalizing state:** Pass control to the next available D.

$Q = D_e$ **in state** $S2$**:** Control is now given to D_e. If D_e was waiting on some label a, close off the current component and grow a new normal component $C(D_e)$ with label a. Otherwise if $C(D_e)$ is already started then we let D_e be now in pending state $S3$. Pass control to the next available D.

$Q = D_e$ **in state** $S3$**:** Control is now given to D_e. We now let D_e be in diagonalizing state $S4$. Freeze $\mathcal{B}$. Close off the currently active component, and grow a new special component $C_0(D_e)$ with label $a+1$ attached to $C(D_e)$.

$Q = D_e$ **in state** $S4$**:** Control is now given to D_e. If $C^*(D_e)$ is not yet defined, we close off the currently active component (which will necessarily be $C_0(D_e)$) and grow $C^*(D_e)$ and $C_0^*(D_e)$. Otherwise if $C^*(D_e)$ is already previously defined, we declare D_e to be now finished $S5$. Unfreeze $\mathcal{B}$ and grow $\mathcal{B}$ differently (as described above). Pass control to the next available D.

If Q is found and given attention, proceed to the next stage of the construction (skipping Phase 2). We make a technical remark here. We do not allow D to close off the current component until every higher priority P requirement in the defining state has responded with a matching component. This does not cause any additional issues as this adds finitely more steps and we can grow the current component while waiting. Otherwise, if no Q is found, D_{e_s} remains in control and we proceed to Phase 2. Notice that for each label a used by the construction, there are at most two components of $\mathcal{A}$ with label a.

Phase 2, growing the currently active component

We define the functions u and α. Let $u_{q,e}(x)$ be the number of steps needed to perform the following computable procedure within the structure $\mathcal{B}_e$:

(i) Evaluate $r_e(x)$. If $r_e(x) \neq q$ we stop successfully.

(ii) Otherwise $r_e(x) = q$. Apply p_e to x to search for some $0 < i \le 2^{e+x+1}$ such that $p_e^i(x)$ has an attached cycle of length $2e+1$. Call this element y_0. If y_0 is not found, we stop *unsuccessfully*. If we reach the root q before 2^{e+x+1} many applications of p_e, we stop *successfully*. (Otherwise assume y_0 is found.)

(iii) Starting from y_0, compute $y_j = p_e^{2^{e+1}}(y_{j-1})$ for $1 \le j \le x+1$. Again, if we reach the root q before computing y_{x+1} we stop successfully. Otherwise $y_1, y_2, \cdots, y_{x+1}$ are all found. Finally if there is at least one y_j for $1 \le j \le x+1$ which has an attached cycle of length $2e+1$, we stop successfully, otherwise we stop unsuccessfully.

This procedure must terminate because all functions evaluated by the procedure are total and all searches are bounded. The number of steps required to complete this procedure is denoted by $u_{q,e}(x)$. The procedure will terminate either successfully or unsuccessfully. Intuitively, an unsuccessful termination allows the construction to kill off P_e by showing $\mathcal{B}_e \not\cong \mathcal{A}$. Note that for each e, the function $(q,x) \mapsto u_{q,e}(x)$ is primitive recursive, but the function $(e,q,x) \mapsto u_{q,e}(x)$ is not primitive recursive in general (although computable). Nevertheless, this function has a primitive recursive graph, i.e. $\{(e,q,x,y) \mid u_{q,e}(x) = y\}$ is a primitive recursive set.

Now we will define the function $\alpha(a, i, \langle e, m\rangle)$. We explain the intended meaning of these parameters. Each component C of the structure $\mathcal{A}$ can be uniquely identified by its label a and whether C is the first component with label a enumerated by the construction ($i = 1$) or if C is the second ($i = 2$). The output of $\alpha(a, i, \langle e, m\rangle)$ will be the length of the cycle attached to the $\langle e, m\rangle$th element of the chain of C. We define α by primitive recursion on m.

- For fixed a, i and e, we let $\alpha(a, i, \langle e, m\rangle) = 2e + 1$ for all m such that $\mathcal{B}_e[m]$ does not yet have a component of type a, i.
- Suppose m_0 is the first such that $\mathcal{B}_e[m_0]$ has a component of type a, i with root q. (If m_0 does not exist then $\alpha(a, i, \langle e, m\rangle) = 2e + 1$ for all m).
- Then for every $m_0 \leq m \leq u_{q,e}(0)$ we define $\alpha(a, i, \langle e, m\rangle) = 2e + 1$. If $u_{q,e}(0)$ stops unsuccessfully we define $\alpha(a, i, \langle e, m\rangle) = 2e + 1$ for all $m > u_{q,e}(0)$.
- Otherwise, $u_{q,e}(0)$ stops successfully. Then for every $u_{q,e}(0) < m \leq u_{q,r}(1)$ we repeat the pattern $2e + 1, 2e + 2$. What this means is that we define $\alpha(a, i, \langle e, u_{q,e}(0) + 2j + 1\rangle) = 2e + 1$ and $\alpha(a, i, \langle e, u_{q,e}(0) + 2j + 2\rangle) = 2e + 2$ for enough j. In order to complete the pattern we assume that $u_{q,e}(1) - u_{q,e}(0)$ is divisible by 2. If $u_{q,e}(1)$ stops unsuccessfully we define $\alpha(a, i, \langle e, m\rangle)$ to repeat the pattern $2e + 1, 2e + 2$ for all $m > u_{q,e}(1)$.
- Otherwise $u_{q,e}(1)$ stops successfully and we proceed with $u_{q,e}(2)$. Generally if some $u_{q,e}(k)$ stops unsuccessfully, we repeat the pattern $2e + 1, \underbrace{2e + 2, \cdots}_{k \text{ times}}$ forever. Otherwise, every $u_{q,e}(k)$ stops successfully. We end up repeating the pattern $2e+1, \underbrace{2e+2, \cdots}_{k \text{ times}}$ for $u_{q,e}(k-1) < m \leq u_{q,e}(k)$. To ensure that the pattern is completed we assume that $u_{q,e}(k) - u_{q,e}(k-1)$ is divisible by $k + 1$.

Now it is not hard to see that the function $\alpha(a, i, \langle e, m\rangle)$ is primitive recursive. This follows from the fact that evaluating the structure $\mathcal{B}_e[m]$ takes bounded many steps, and that the graph of u is primitive recursive. At a particular step of the recursion, if $m > u_{q,e}(k-1)$, we cannot primitive recursively evaluate $u_{q,e}(k)$, but we do not need to as we only need to check if $u_{q,e}(k) = m$.

Having obtained the primitive recursive function α we can now describe what the construction does in Phase 2 at stage s. Suppose the currently active component (which we want to keep growing) has label a, root x and is the ith component with label a. Suppose n is the largest such that $s^n(x)\uparrow$. We define $s^n(x)$ to be the next element of the domain and primitively recursively grow an attached cycle of length $\alpha(a, i, n)$. Define p, c and r appropriately. This ends Phase 2.

4.3.3. Verification

By the construction, at every stage s, exactly one D_{e_s} is in control, and is in state $S2$ or $S4$. At each stage of the construction, everything we evaluate is time-bounded. Thus, $\mathcal{A}$ is a primitive recursive structure. No D is in state $S4$ forever, hence $\mathcal{B} \cong \mathcal{A}$. It now remains to check that the requirements are met. It is clear that each requirement receives attention and is initialized finitely often.

Lemma 4.4. *Each P_e is satisfied.*

Proof. Assume that $\mathcal{B}_e \cong \mathcal{A}$ and we fix a stage after which P_e is never initialized. First of all we claim that P_e is never placed in the diagonalizing state. Suppose this happens at some stage s_0. The active $\mathcal{B}_e$-component $C^{\mathcal{B}_e}$ is declared the diagonalizing component at s_0. At s_0, one of the following must hold of $C^{\mathcal{B}_e}$:

(i) The label of $C^{\mathcal{B}_e}$ is larger than any existing label in $\mathcal{A}$ (and has not yet converged).
(ii) The label of $C^{\mathcal{B}_e}$ has converged but is different from existing labels in $\mathcal{A}$ (taking multiplicity into account).
(iii) Some cycle attached to $C^{\mathcal{B}_e}$ is of a different length than the corresponding cycle in $\mathcal{A}$.
(iv) The chain of $C^{\mathcal{B}_e}$ is longer than the length of the chain in the corresponding component in $\mathcal{A}$ (which is closed).
(v) $\mathcal{B}_e$ has more root elements than $\mathcal{A}$.
(vi) The functions p_e, s_e, r_e, c_e do not map correctly, for instance, r_e maps to a non-root element.

In (vi) clearly $\mathcal{B}_e \ncong \mathcal{A}$ just because the structure is not of the correct type. In (iii) (iv) and (v) the component $C^{\mathcal{B}_e}$ has no matching image in $\mathcal{A}$; note that any pair of components in $\mathcal{A}$ with the same label are identical. In the case of (ii) no new component with the same label as $C^{\mathcal{B}_e}$ can be later introduced as all lower priority D are initialized and higher priority D are assumed not to act any more. In the case of (i) all new components in $\mathcal{A}$ are introduced by some D of lower priority, and due to the wait by D in state $S2$ will have new label smaller than the (eventual) label of $C^{\mathcal{B}_e}$. If the label of $C^{\mathcal{B}_e}$ eventually converges, we initialize all lower priority D and if not then $\mathcal{B}_e \ncong \mathcal{A}$ as all labels in $\mathcal{A}$ are finite.

Thus P_e is always in the defining state. We now describe how to define primitive recursive isomorphisms $h_e : \mathcal{B}_e \mapsto \mathcal{A}$ and $\hat{h}_e : \mathcal{A} \mapsto \mathcal{B}_e$. It suffices to define h_e and $\hat{h}_e$ on root elements (recognizing if a given element is a root element requires only a single application of r or r_e), and to argue that $\chi_{\mathcal{B}_e}$ is primitive recursive. Here $\chi_{\mathcal{B}_e}(x)$ is the coordinates (n, m) of element $x \in \mathcal{B}_e$ in its component.

First we define $\hat{h}_e(q)$ for a root element $q \in \mathcal{A}$. We run the construction until the stage where C_q is introduced into the construction; this takes at most q construction steps. As only lower priority Ds are active, no new component is introduced in $\mathcal{A}$ before P_e closes off the old component and introduces a new matching $C_q^{\mathcal{B}_e}$ in $\mathcal{B}_e$. So by applying s_e at most q times to the structure $\mathcal{B}_e[q]$ and r_e once we will be able to find the root of the matching component $C_q^{\mathcal{B}_e}$ in $\mathcal{B}_e$. Notice that if C_q is of the type $C^*(D_i)$ or $C_0^*(D_i)$ for some i, as we grow both components simultaneously in $\mathcal{A}$, we can force P_e to reveal both matching components in $\mathcal{B}_e$ before matching.

Now we describe how to define $h_e(q)$ for a root element $q \in \mathcal{B}_e$. Run the construction until a stage s where the construction recognizes that q is a root element of $\mathcal{B}_e$ (this takes as many steps as to evaluate $r_e(q)$). Let C_u be the current active component of $\mathcal{A}$ at stage s. If $h_e^{-1}(u)$ is already defined then at stage s the construction sees that $\mathcal{B}_e[s]$ (having more components) is not a substructure of $\mathcal{A}[s]$ and thus will put P_e in the diagonalizing state with diagonalizing component q. Thus $h_e^{-1}(u)$ is not defined. Also u must be the only root element of $\mathcal{A}$ which is not in the range of h_e because otherwise $\mathcal{A}[s]$ has at least two more components than $\mathcal{B}_e[s]$, which is impossible since any D will wait for P_e to catch up before closing a component. Hence the components of u and q must have the same label (otherwise P_e gets into the diagonalizing state), and thus be matching. We define $h_e(q) = u$.

Notice that if there are two roots with the same label, then h_e and $\hat{h}_e$ have to map the root of the first component to the root of the first component.

Finally we show that $\chi_{\mathcal{B}_e}$ is primitive recursive. Given $x \in \mathcal{B}_e$. Compute $q = r_e(x)$ and $u = h_e(q)$. Note that $p_e(s_e(x)) = r_e(x)$ if and only if $x = q$ or $x \in Y_q^0$. In this case we know that the n-coordinate of x is 0. To figure out the m-coordinate of x, notice that the label of u is at most u. So the m-coordinate of x can be found by applying c_e to x at most u many times.

Now we assume that $p_e(s_e(x)) \neq r_e(x)$, that is, the n-coordinate of x is positive. If we are given the n-coordinate of x, we can (primitively recursively) compute the m-coordinate of x because the cycle containing x is of length $2e+1$ or $2e+2$, where $n = \langle e, j\rangle$. It therefore suffices to give a fast procedure to compute the n-coordinate of x.

First of all, recall that for fixed e, the function $u_{q,e}(k)$ is primitive recursive. We claim that $u_{q,e}(k)$ stops successfully for all k. Suppose not, and that k is the least such that $u_{q,e}(k)$ stops unsuccessfully. (Note that q and u are both associated with the same pair (a, i).) In the component C_u we will end up repeating the pattern $2e+1, \underbrace{2e+2}_{k \text{ times}}$ as the cycle length attached to the $\langle e, j\rangle$th element of the chain, for $j > u_{q,e}(k-1)$. This means that the longest consecutive run of js where $s_e^{\langle e,j\rangle}(q)$ has an attached cycle of length $2e+2$ is k.

There are two reasons why $u_{q,e}(k)$ stops unsuccessfully: Either y_0 is not found or all of $y_1, \cdots, y_{k+1}$ are attached a cycle of length $2e+2$. We get a contradiction in either case.

Thus $u_{q,e}(k)$ stops successfully for all k. We claim that the n-coordinate of x cannot be more than $u_{q,e}(x)$. Suppose it is. In that case, x is close to a sequence $s_e^{\langle e,j\rangle}(q), s_e^{\langle e,j+1\rangle}(q), \cdots, s_e^{\langle e,j+j'\rangle}(q)$ where $s_e^{\langle e,j\rangle}(q)$ and $s_e^{\langle e,j+j'\rangle}(q)$ have a cycle of length $2e+1$, and all other terms have a cycle of length $2e+2$. Here $j'-1 \leq x$ is the consecutive number of $2e+2$ cycles. This number is too small as in C_u we will repeat with longer runs of $2e+2$ after $u_{q,e}(k)$. □

Lemma 4.5. *Each D_e is satisfied.*

Proof. Fix a stage after which D_e is never initialized. The state of D_e cannot decrease after it is never initialized. Clearly each D spends a finite amount of time in states $S2$ and $S4$ each time, hence, no D retains control for cofinitely many stages. This means that the final state of D_e cannot be $S1$. Hence the final state of D_e has to be $S3$ or $S5$. If the final state is $S3$ then φ_e is not an isomorphism. Let's consider now that the final state is $S5$. That means that at the end of state $S4$ when we unfreeze $\mathcal{B}$ we would have attached a special component with label $a+2$ to $C(D_e)^{\mathcal{B}}$. Since φ_e maps $C(D_e)^{\mathcal{B}}$ to $C(D_e)^{\mathcal{A}}$, this means that φ_e cannot be an isomorphism. □

This concludes the proof of Theorem 1.8. □

An easy generalization of the proof of Theorem 1.8 yields the following:

Corollary 4.6. *There is a fpr structure $\mathcal{A}$ and an existential unary predicate Q in the language of $\mathcal{A}$ such that Q is primitive recursive in all fpr copies of $\mathcal{A}$ and for some computable copy $\mathcal{B} \cong \mathcal{A}$ the predicate Q is not decidable.*

Proof. The trick is in making $\mathcal{A}$ fpr-categorical and combining the copying strategies from Theorem 1.8 with a new diagonalization strategy making sure that the predicate is not computable in $\mathcal{B}$. We give more detail.

We will construct a structure $\mathcal{A}$ in the same language and with the same properties as in Theorem 1.8. There is one modification needed. We introduce a new type of component called a *red component*. A red component is consists of a single element, the root x, and where $s(x) = r(x) = c(x) = x$. This component is a special component, and will be attached to a normal component of the construction. That is, $p(x)$ will be the root of some normal component. A red component is attached to every normal component of the form $C^*(D_e)$ (and only these ones). Hence $C^*(D_e)$ will have a red component c_r attached to it, and the corresponding $C_0^*(D_e)$ will be attached to c_r (instead of being attached to $C^*(D_e)$ directly).

The predicate Q is defined by the formula

$$r(x) = x \,\&\, (\exists z \neq x)[r(z) = z \,\&\, p(z) = x \,\&\, c(z) = z],$$

which asserts that x is a root element with an attached red component.

In $\mathcal{A}$ the set $Q^{\mathcal{A}}$ is primitive recursive, since given an element x we can check if it is a root, and if so, compute the construction up till the point where x is introduced (this happens by stage x of the construction). Check if there is already an existing component with the same label as x. If so then $x \in Q^{\mathcal{A}}$, otherwise $x \notin Q^{\mathcal{A}}$.

Now if $Q^{\mathcal{B}}$ is computable we claim that there is a computable isomorphism $\varphi : \mathcal{B} \mapsto \mathcal{A}$. Fix a computable enumeration of $\mathcal{A}$ and $\mathcal{B}$. It suffices to define φ on root elements correctly, since each component itself is rigid. Given a root element $z \in \mathcal{B}$, we first compute the label of $C_z^{\mathcal{B}}$ (say, it is a) and check if $z \in Q^{\mathcal{B}}$. If yes, then we enumerate $\mathcal{A}[s]$ until the second component with label a is revealed in $\mathcal{A}$. If no, then we can map z to the root of the first component with label a in $\mathcal{A}$. □

References

[1] P.E. Alaev, Existence and uniqueness of structures computable in polynomial time, Algebra Logic 55 (1) (2016) 72–76.
[2] Uri Andrews, Decidable models of ω-stable theories, J. Symbolic Logic 79 (1) (2014) 186–192.
[3] C. Ash, J. Knight, Computable Structures and the Hyperarithmetical Hierarchy, Stud. Logic Found. Math., vol. 144, North-Holland Publishing Co., Amsterdam, 2000.
[4] Gábor Braun, Lutz Strüngmann, Breaking up finite automata presentable torsion-free Abelian groups, Internat. J. Algebra Comput. 21 (8) (2011) 1463–1472.
[5] D. Cenzer, J.B. Remmel, Polynomial time versus computable Boolean algebras, in: M. Arslanov, S. Lempp (Eds.), Recursion Theory and Complexity, Proceedings 1997 Kazan Workshop, de Gruyter, 1999, pp. 15–53.
[6] Douglas Cenzer, Rodney G. Downey, Jeffrey B. Remmel, Zia Uddin, Space complexity of Abelian groups, Arch. Math. Logic 48 (1) (2009) 115–140.
[7] Douglas A. Cenzer, Jeffrey B. Remmel, Polynomial-time versus recursive models, Ann. Pure Appl. Logic 54 (1) (1991) 17–58.
[8] Douglas A. Cenzer, Jeffrey B. Remmel, Polynomial-time Abelian groups, Ann. Pure Appl. Logic 56 (1–3) (1992) 313–363.
[9] R. Dedekind, Was sind und was sollen die Zahlen?, 8te unveränderte Aufl, Friedr. Vieweg & Sohn, Braunschweig, 1960.
[10] R. Downey, On presentations of algebraic structures, in: Complexity, Logic, and Recursion Theory, in: Lect. Notes Pure Appl. Math., vol. 187, Dekker, New York, 1997, pp. 157–205.
[11] R. Downey, Computability theory and linear orderings, in: Handbook of Recursive Mathematics, vol. 2, in: Stud. Logic Found. Math., vol. 139, North-Holland, Amsterdam, 1998, pp. 823–976.
[12] David B.A. Epstein, James W. Cannon, Derek F. Holt, Silvio V.F. Levy, Michael S. Paterson, William P. Thurston, Word Processing in Groups, Jones and Bartlett Publishers, Boston, MA, 1992.
[13] Y. Ershov, S. Goncharov, Constructive Models, Siberian School of Algebra and Logic, Consultants Bureau, New York, 2000.
[14] Yu. Ershov, Problems of Solubility and Constructive Models, Nauka, Moscow, 1980 (in Russian).
[15] Ekaterina B. Fokina, Iskander Sh. Kalimullin, Russell Miller, Degrees of categoricity of computable structures, Arch. Math. Logic 49 (1) (2010) 51–67.
[16] G.A. Freiman, Nachala strukturnoi teorii slozheniya mnozhestv, Kazan. Gosudarstv. Ped. Inst., Elabuž. Gosudarstv. Ped. Inst., Kazan, 1966.
[17] L. Fuchs, Infinite Abelian Groups, vol. I, Pure Appl. Math., vol. 36, Academic Press, New York, 1970.
[18] S. Goncharov, The problem of the number of nonautoequivalent constructivizations, Algebra Logika 19 (6) (1980) 621–639, 745.
[19] S. Goncharov, Countable Boolean Algebras and Decidability, Siberian School of Algebra and Logic, Consultants Bureau, New York, 1997.
[20] S. Goncharov, S. Lempp, R. Solomon, The computable dimension of ordered Abelian groups, Adv. Math. 175 (1) (2003) 102–143.
[21] Sergey Goncharov, Bakhadyr Khoussainov, Open problems in the theory of constructive algebraic systems, in: Computability Theory and Its Applications, Boulder, CO, 1999, in: Contemp. Math., vol. 257, Amer. Math. Soc., Providence, RI, 2000, pp. 145–170.
[22] Serge Grigorieff, Every recursive linear ordering has a copy in dtime-space(n, log(n)), J. Symbolic Logic 55 (1) (1990) 260–276.
[23] Leo Harrington, Recursively presentable prime models, J. Symbolic Logic 39 (2) (1974) 305–309.
[24] M. Harrison-Trainor, A. Melnikov, A. Montalban, Independence in computable algebra, J. Algebra 443 (2015) 441–468, http://dx.doi.org/10.1016/j.jalgebra.2015.06.004.
[25] G. Higman, Subgroups of finitely presented groups, Proc. R. Soc. Ser. A 262 (1961) 455–475.
[26] Denis R. Hirschfeldt, Bakhadyr Khoussainov, Richard A. Shore, A computably categorical structure whose expansion by a constant has infinite computable dimension, J. Symbolic Logic 68 (4) (2003) 1199–1241.
[27] N. Khisamiev, Constructive Abelian groups, in: Handbook of Recursive Mathematics, vol. 2, in: Stud. Logic Found. Math., vol. 139, North-Holland, Amsterdam, 1998, pp. 1177–1231.
[28] Bakhadyr Khoussainov, Anil Nerode, Automatic presentations of structures, in: Logical and Computational Complexity. Selected Papers. Logic and Computational Complexity, International Workshop LCC '94, Indianapolis, Indiana, USA, 13–16 October 1994, 1994, pp. 367–392.
[29] Bakhadyr Khoussainov, Anil Nerode, Open questions in the theory of automatic structures, Bull. Eur. Assoc. Theor. Comput. Sci. EATCS 94 (2008) 181–204.
[30] Bakhadyr Khoussainov, André Nies, Sasha Rubin, Frank Stephan, Automatic structures: richness and limitations, Log. Methods Comput. Sci. 3 (2:2) (2007) 1–18.
[31] Khoussainov Bakhadyr, Richard A. Shore, Effective model theory: the number of models and their complexity, in: Models and Computability, Leeds, 1997, in: London Math. Soc. Lecture Note Ser., vol. 259, Cambridge Univ. Press, Cambridge, 1999, pp. 193–239.
[32] P. LaRoche, Recursively presented Boolean algebras, Notices Amer. Math. Soc. 24 (1977) 552–553.
[33] A. Mal'cev, Constructive algebras. I, Uspekhi Mat. Nauk 16 (3 (99)) (1961) 3–60.
[34] Alexander G. Melnikov, Computable Abelian groups, Bull. Symbolic Logic 20 (3) (2014) 315–356.
[35] Terrence S. Millar, Pure recursive model theory, in: Handbook of Computability Theory, in: Stud. Logic Found. Math., vol. 140, North-Holland, Amsterdam, 1999, pp. 507–532.
[36] Russell Miller, An introduction to computable model theory on groups and fields, Groups Complex. Cryptol. 3 (1) (2011) 25–45.
[37] Russell Miller, Hans Schoutens, Computably categorical fields via Fermat's last theorem, Computability 2 (1) (2013) 51–65.
[38] André Nies, Pavel Semukhin, Finite automata presentable Abelian groups, in: Logical Foundations of Computer Science, in: Lecture Notes in Comput. Sci., vol. 4514, Springer, Berlin, 2007, pp. 422–436.
[39] André Nies, Richard M. Thomas, FA-presentable groups and rings, J. Algebra 320 (2) (2008) 569–585.
[40] A. Nurtazin, Computable Classes and Algebraic Criteria of Autostability, Summary of Scientific Schools, Math. Inst. SB USSRAS, Novosibirsk, 1974.
[41] M. Rabin, Computable algebra, general theory and theory of computable fields, Trans. Amer. Math. Soc. 95 (1960) 341–360.

[42] J. Remmel, Recursive Boolean algebras, in: J.D. Monk (Ed.), Handbook of Boolean Algebras, North-Holland Publishing Co., 1989, pp. 1099–1165, Ch. 25.
[43] Jeffrey B. Remmel, Recursive Boolean algebras with recursive atoms, J. Symbolic Logic 46 (3) (1981) 595–616.
[44] S. Simpson, Subsystems of Second Order Arithmetic, second edition, Perspect. Log., Cambridge University Press, Cambridge, 2009.
[45] T. Slaman, Relative to any nonrecursive set, Proc. Amer. Math. Soc. 126 (7) (1998) 2117–2122.
[46] R. Smith, Two theorems on autostability in p-groups, in: Logic Year 1979–80, Proc. Seminars and Conf. Math. Logic, Univ. Connecticut, Storrs, Conn., 1979/80, in: Lecture Notes in Math., vol. 859, Springer, Berlin, 1981, pp. 302–311.
[47] R. Soare, Recursively Enumerable Sets and Degrees: A Study of Computable Functions and Computably Generated Sets, Persp. Math. Logic, Springer-Verlag, Berlin, 1987.
[48] Tsankov Todor, The additive group of the rationals does not have an automatic presentation, J. Symbolic Logic 76 (4) (2011) 1341–1351.
[49] B. van der Waerden, Eine Bemerkung über die Unzerlegbarkeit von Polynomen, Math. Ann. 102 (1) (1930) 738–739.
[50] S. Wehner, Enumerations, countable structures and Turing degrees, Proc. Amer. Math. Soc. 126 (7) (1998) 2131–2139.