A STRUCTURE OF PUNCTUAL DIMENSION TWO

ALEXANDER MELNIKOV AND KENG MENG NG

Abstract. The paper contributes to the general program which aims to eliminate unbounded search from proofs and procedures in computable structure theory. A countable structure in a finite language is punctual if its domain is omega and its operations and relations are primitive recursive. A function f is punctual if both f and f^{-1} are primitive recursive. We prove that there exists a countable rigid algebraic structure which has exactly two punctual presentations, up to punctual isomorphism. This solves a problem left open in [Mel17]; see also [BDKM].

1. Introduction

After decades of development, computability theory and computable structure theory [EG00, AK00] gave a well-developed framework to investigate the limits of computation in mathematics. Beginning in the 1980's and rather independently, there has been quite a lot of work on online infinite combinatorics; see [Kie81, Kie98, KPT94, LST89, Rem86]. Nonetheless, there is no general and established theory for online structures, and until recently there has been very little (if any) correlation between computable structure theory and online combinatorics. The paper contributes to a new general program [KMN17b, Mel17, BDKM, KMN17a, MN] that aims to lay the foundations of online computability in algebra and combinatorics. The new program has many aspects; see surveys [Mel17, BDKM] for a detailed exposition. The main result of the paper belongs to a branch of this new framework which is motivated by the classical results on (Turing) computable algebraic structures. The result resembles the well-known theorem of Goncharov [Gon80, Gon81] saying that there is a structure of computable dimension two. Informally, our theorem says that there is a structure of "online" or "punctual" dimension two; the formal definitions will be given shortly. Although the statement of our result is similar to the statement of the above mentioned Goncharov's theorem, our proof shares almost nothing in common with the proof in [Gon80] (or with any other known dimension two proof). Now to the details.

1.1. Turing computable mathematics.

1.1. Turing computable mathematics. The general area of *computable* or *effective mathematics* is devoted to understanding the algorithmic content of mathematics. The roots of the subject go back to the beginning of the 20th century as discussed in [MN82]. Early work concentrated on developing algorithmic mathematics in algebra, analysis, and randomness; e.g., [Her26, Deh11, Bor09, vM19]. These ideas were later recast by Godel, Kleene, Church, Turing and others and, in particular, lead to the refutation of Hilbert's conjecture on decidability of first order logic.

The standard model for such investigations is a (Turing) *computable presentation* of a structure. By this we mean a coding of the structure with universe $\mathbb{N}$, and the relations and functions coded Turing computably. There has been a large body of work on Turing computable presentations of structures, see books [EG00, AK00] and the relatively recent surveys [FHM14, Mil11].

One topic in such investigations is the study of computable structures up to computable isomorphism. The motivation here is clear: algebraic groups and fields are viewed up to algebraic isomorphism, topological groups and rings are studied up to algebraic homeomorphism, and therefore the right morphisms in the category of *computable* algebraic structures are the *computable* algebraic isomorphisms. Maltsev [Mal61] was perhaps the first to make this idea explicit and formal. He also initiated a systematic study of structures which have a unique (Turing) computable presentation up to (Turing) computable isomorphism. Such structures are called *computably categorical* or *autostable*. As was first noted by Goncharov, in many natural classes an algebraic structure has either exactly one or infinitely many computable presentations up to computable isomorphism, see [EG00] for many results illustrating this

This work is supported by the Marsden Foundation of New Zealand.

dichotomy. Remarkably, via an intricate argument Goncharov [Gon80] constructed an algebraic structure which has *exactly two* computable presentations up to computable isomorphism. Although the first such structure was algebraically artificial, similar examples can be found among two step nilpotent groups [Gon81], (remarkably) fields [MPSS18], and some other natural classes [HKSS02]. There has been many further remarkable works on finite computable dimension with applications to degree spectra of relations and categoricity spectra; see the somewhat dated survey [KS99], the excellent PhD thesis of Hirschfeldt [Hir99], and also the very recent paper [CS19].

Note that this framework uses the general notion of a Turing computable function. In particular, we put no resource bound on our computation. One therefore naturally seeks to understand whether the abstract algorithms from computable structure theory can be made more feasible.

1.2. **Feasible mathematics.**

What happens when we put resource bounds on the definitions of allowable computation?

Khoussainov and Nerode [KN94] initiated a systematic study into *automatically presentable* algebraic structures. Automatic structures are linear-time computable and have decidable theories, but such presentations seem quite rare. For example, the additive group of the rationals is not automatic [Tsa11]. The approach via finite automata is highly sensitive to how we define what we mean by automatic. See [ECH$^+$92] for an alternate approach to automatic groups. Cenzer and Remmel, Gregorieff, Alaev, and others [CR98, Gri90, Ala17, Ala18] studied the much more general notion *polynomial time* presentable structures. We omit the formal definitions, but we note that they are sensitive to how exactly we code the domain. In contrast with automatic structures, in many common algebraic classes we can show that each Turing computable structures has a polynomial-time computable isomorphic copy [Gri90, CR, CR92, CDRU09, CR91].

Kalimullin, Melniukov and Ng [KMN17b] noted that many known proofs from polynomial time structure theory (e.g., [CR91, CR92, CDRU09, Gri90]) are focused on making the operations and relations on the structure merely *primitive recursive*, and then observing that the presentation that we obtain is in fact polynomial-time. The restricted Church-Turing thesis for primitive recursive functions says that a function is primitive recursive iff it can be described by an algorithm that uses only bounded loops. Furthermore, to illustrate that a structure has no polynomial time copy, it typically easiest to argue that it does not even have a copy with primitive recursive operations; see e.g. [CR92]. From a computable model theory perspective, the most natural way to construct structures is via a Henkin construction. Almost all natural decision procedures in the literature are primitive recursive, and as observed in [KMN17b] the natural Henkin construction will automatically give an appropriately primitive recursively decidable model.

Kalimullin, Melnikov and Ng [KMN17b] thus proposed that primitive recursive structures provide an adequate and rather general model to unite the theories of feasible (polynomial-time) structures and online combinatorics. Although their approach may seem way too general, they very shortly discovered that "merely" forbidding unbounded search leads to a profound impact on both intuition and techniques. Also, compare this to the approach in, e.g., Kierstead [Kie81] where the only restriction on an algorithm is that it must have a bounded use, and there is *no resource bound* imposed otherwise.

Primitive recursiveness gives a useful *unifying abstraction* to computational processes for structures with computationally bounded presentations. In such investigations we only care that there is *some* bound. We have to act "now" or "without unbounded search", where these notions are formalised in the sense that we can precompute the bound. Irrelevant counting combinatorics is often stripped off proofs thus emphasising the effects related to the existence of a bound in principle. These effects are far more significant than it may seem at first glance; the non-trivial and novel proof of the main result of the paper will be a good illustration of this phenomenon. See [BDKM] for a detailed exposition of the new unexpectedly rich and technically non-trivial emerging theory of primitive recursive structures. Below we focus only on the aspects of the theory relevant to the present article.

1.3. **Punctual computability.**

Kalimullin, Melnikov, and Ng [KMN17b] proposed that an "online" structure must *minimally* satisfy:

Definition 1.1 ([KMN17b])**.** A countable structure is *fully primitive recursive* (fpr) if its domain is $\mathbb{N}$ and the operations and predicates of the structure are (uniformly) primitive recursive.

We call fpr structures *computable without delay*, *punctually computable*, or simply *punctual*. The intuition is that a punctual structure must reveal itself without unspecified delay. Here "delay" really means an instance of a truly unbounded search. We could also agree that all finite structures are also punctual by allowing initial segments of $\mathbb{N}$ to serve as their domains. Although the definition above is not restricted to finite languages, we will never consider infinite languages in the paper; therefore, we do not clarify what uniformity means in Def. 1.2.

Recall that the inverse of a primitive recursive function does not have to be primitive recursive.

Definition 1.2 ([KMN17b]). A function $f : \mathbb{N} \to \mathbb{N}$ is punctual if both f and f^{-1} are primitive recursive.

Punctual isomorphisms appear to be the most natural morphisms in the category of punctual structures. We say that a structure is *punctually categorical* if it has a unique punctual presentation up to punctual isomorphism.

Kalimullin, Melnikov and Ng [KMN17b] characterised punctual categoricity in many standard algebraic classes. Similarly to the above-mentioned "1 vs. ω" Goncharov's dichotomy in (Turing) computability, it is easy to see that in each of these classes a structure has either one or infinitely many punctual copies. Thus one naturally seeks to either confirm or refute the conjecture saying that the 1 vs. ω dichotomy holds in the punctual world. The main obstacle in proving or disproving the conjecture has been the lack of adequate techniques and, more importantly, of intuition. To illustrate the counter-intuitive nature of punctual structures, we mention that Kalimullin, Melnikov and Ng [KMN17b] constructed a punctually categorical structure which is not computably categorical. Although this sounds contradictory, the latter does not (formally) imply the former; nonetheless, all natural examples seemed to confirm that the implication must hold. After a few years of investigation, we have finally accumulated enough intuition and technical tools to finally refute the conjecture.

Theorem 1.3. *There exists an algebraic structure which has exactly two punctual presentations, up to punctual isomorphism.*

The proof combines a "pressing" strategy from [KMN17b] with a new technique and several new ideas. We emphasise that our proof shares virtually nothing in common with Goncharov's dimension two proof. The only similarity is perhaps their hight combinatorial complexity. The reader therefore should prepare themselves for a relatively involved proof. We strongly conjecture that our proof can be modified to produce a structure with exactly n incomparable degrees. We however leave it as a conjecture for someone to verify. We also believe that both the result and the new techniques introduced in its proof will find important applications in the theory of punctual structures, and perhaps beyond.

The rest of the paper is devoted to the proof of Theorem 1.3. We also state several further open questions in a short conclusion (Section 3).

2. Proof of Theorem 1.3

2.1. **The requirements.** We are building two punctual presentations $\mathcal{A} \cong \mathcal{B}$ of a countably infinite *rigid* structure in a finite language which will be described in due course.

We need to meet the following requirements:

$$\mathcal{A}|_{pr}\mathcal{B}$$

and

$$P_e \cong \mathcal{A} \implies P_e \cong_{pr} \mathcal{A} \text{ or } P_e \cong_{pr} \mathcal{B},$$

where P_e stands for the e'th punctual presentation in their total enumeration (recall the domain has to be the whole of ω).

The former requirement we split into subrequirements:

$$p_e : \mathcal{A} \not\to_{\cong} \mathcal{B} \text{ and } p_e : \mathcal{B} \not\to_{\cong} \mathcal{A},$$

where p_e stands for the e'th primitive recursive function in their uniform total enumeration.

2.2. **The pressing strategy.**

Although these below requirements are not quite the requirements we will have to meet, it is easier to describe the strategy (which we call *the pressing strategy*) for these simplified requirements; we then modify the strategy to alternate between two copies.

Thus, we attempt to meet, for each e, the requirement

$$\mathcal{A} \cong P_e \implies \mathcal{A} \cong_{fpr} P_e,$$

where $(P_e)_{e\in\omega}$ is the natural uniformly computable listing of all punctual structures. Clearly, the list itself is not primitive recursive, for otherwise we would be able to produce a punctual structure which is not in the list. The reader should think of P_e as of being "increasingly slow" in e. However, we will argue that for each fixed e there is a primitive recursive time-function, i.e., a function that bounds the speed of approximation of $P_e = \bigcup_s P_{e,s}$ within the overall uniform primitive recursive approximation $(P_{e,s})_{e,s\in\omega}$. We take this property for granted throughout the proof; see the Appendix of [BDKM] for a formal clarification.

2.2.1. *Pressing P_0.*

The idea is as follows. Start by building an infinite chain using a unary function S:

$$0 \to S(0) \to S^2(0) \to S^3(0) \to \cdots,$$

and use another unary function, say U, to attach a U-loop of some fixed small size to each S^n. To be more specific, suppose we attach 2-loops. Use another unary function r that sends each point back to the origin:

$$\forall x\, r(x) = 0.$$

Do nothing else and wait for the opponent's structure P_0 to respond. (The structure will be rigid.)

The opponent's structure P_0 must give us a few 2-loops, otherwise $P_0 \not\cong \mathcal{A}$. However, it is important to see *how exactly* P_0 could fail to be isomorphic to $\mathcal{A}$.

(1) The structure P_0 does not even look right; that is, it is not an S-chain, etc. In this case we do nothing.
(2) Otherwise, P_0 could give us an U-loop of a wrong size, say 4. Then we will forever forbid 4 in the construction.
(3) P_0 starts growing a long simple U-chain. It is easiest to drive it to infinity in the construction, as follows. At stage s other strategies will be allowed to use only loops that are shorter than the U-chain as seen in $P_0[s]$.

Assume none of the above cases apply. Then P_0 does respond by giving us a few consequent 2-loops. *Note that, perhaps P_0 has not revealed the position of x in the S-chain.* This point could correspond to one of the 2-loops that we have produced in $\mathcal{A}$ while waiting for the slow P_0 to give us something. But one of our tasks is to demonstrate that the (unique) primitive recursive isomorphism from P_0 to $\mathcal{A}$ is primitive recursive. We must decide promptly where x must be mapped. But we cannot just keep building 2-loops in $\mathcal{A}$, as it will give the opponent an upper hand.

Instead, as soon as P_0 responds by giving a 2-loop, we switch from the pattern

$$2-2-2-2-2-2-2-2-\cdots$$

to the pattern (say)

$$2-4-2-4-2-4-2-4-\cdots,$$

assuming that 4 is currently not forbidden in the construction.

How do we punctually map $x \in P_0$ (see above) to $\mathcal{A}$? Recall that x was a part of a chain of a few consequent 2-loops. In $\mathcal{A}$, the initial segment consisting of adjacent 2-loops has a specific length that we know at the stage, say k. In P_0, calculate r on x to find the origin, and then calculate S of the origin at most k times to figure the position of x.

To compute the unique isomorphism from $\mathcal{A}$ to P_0, simply start from the origin in P_0 and map $\mathcal{A}$ onto P_0 naturally, according to the speed of P_0. We use *the primitive recursive time* which measures the speed of its enumeration (see the discussion above). In a way, this will be a non-uniformly primitive recursive proof.

2.2.2. *Pressing P_0 and P_1.* For simplicity, the highest priority structure can be pressed using loops attached to even positions in the S-chain:

$$0, S^2(0), S^4(0), \cdots, S^{2k}(0), \cdots$$

and the lower priority P_1 will be associated with odd positions of the S-chain. Also, P_0 will be using U-loops of even length, and P_1 of odd length.

The difference is that the loops corresponding to P_0 are located at even positions:

$$2 - \Box - 2 - \Box - 2 - \Box - 2 - \cdots,$$

where the content of the $\Box$s does not worry the strategy. The strategy then switches to:

$$\cdots - 2 - \Box - 4 - \Box - 2 - \Box - 4 - \cdots,$$

assuming 4 is small enough and is not restrained. If the strategy for P_0 must act again then we could sue a more complex pattern of 2 and 4, such as:

$$\cdots - 2 - \Box - 4 - \Box - 4 - \Box - 2 - \Box - 4 - \Box - 4 \cdots.$$

Alternatively, we could start using $2^3 = 8$:

$$\cdots - 2 - \Box - 8 - \Box - 2 - \Box - 8 - \Box - 2 - \Box - 8 \cdots.$$

We prefer to go with the second option. For simplicity, we associate each P_i with loops of sizes p_i^k, $k \in \omega$, where $(p_i)_{i\in\omega}$ is the standard list of all primes. Then we could slowly introduce longer loops into the construction, when we are ready. This will allow to keep the strategies fairly independent from each other.

Remark 2.1. *Even in the most general case of many strategies, we could get around with using* only *2 and 4 (and their patterns) throughout the construction, for P_0. At a late enough stage we will know the size of the interval that we have to check in P_0 to understand where the respective location is.*

From the perspective of the strategy working with P_0, the following scenaria are possible:

- P_0 has an obviously wrong isomorphism type. This is an instant win which requires no action.
- P_0 shows and S-interval of length 4 with a loop of size 2^k, where k has not yet been used. Then the strategy forever forbids the interval, and thus guarantees P_0 is not isomorphic to the structure.
- P_0 shows an S-interval of length 4, and at least one of the loops that could potentially have length of the form 2^k has not closed yet. We wait until the chain grows longer than the largest loop of the form 2^k used so far. While we are waiting, we keep building our chain using the same pattern as before. We will never switch to a new pattern of powers of 2. Again, P_0 must have a wrong isomorphism type.

Remark 2.2. Note that, in the third clause we do not worry about the $\Box$ components, and this trick removes some tensions in the construction. We elaborate it using an example. Suppose we see a sequence $x - y - z - w$ of S-successors. We start evaluating the unary function for all of them. Suppose the P_0-strategy have used loops of sizes 2, 4, 8.

Evaluate the unary function on x, y, z, w exactly 8 times. We have the following sub-cases:

- We discover that exactly two loops (x and z or y and w) form an admissible pattern of powers of 2. Then ignore what happens at the other two points, even if their chains have not yet closed.
- We discover that $x - y - z - w$ cannot possibly give us a right pattern of powers of 2. This is judged based on the sizes of the (finite) loops that we discover.
- None of the two cases above. This means that some of the chains are longer than 8, which makes $k > 3$ in any configuration of the form $2 - \Box - 2^k$ or $2^k - \Box - 2$ that could potentially be realised by the sequence in the future. In this case it is sufficient to forbid 2^k when (and if) it is every discovered. Meanwhile, keep using only $2, 4$, and 8.

Meanwhile, the locations reserved for P_1 – these are marked with $\square$ above – will be filled with 3 and perhaps (later) with 3^k for some k. Our punctual definition of the isomorphism between P_1 and $\mathcal{A}$ is essentially the same as in the description of one strategy in isolation. We only need to look at a bit larger interval in P_1 around a given point x.

2.2.3. *Pressing all P_e at once with a single S-chain.* In the general case of many P_e we generalise the ideas described above. At later stages the construction will respect more of the P-structures. To implement the above idea, we allow P_0 to play at every second location, P_1 at every forth, P_2 at every eighth etc., and we fill the missing locations with loops of size 1. When we are ready to monitor P_j, we start replacing the filler 1-loops with loops of the form p_j^k. This way there is always some room left for the next P_j when it steps into the construction.

Also, using 1 as a filler will allow for a similar analysis as above, where we could pick an interval and challenge the opponent's structure to show us loops or chains in the interval. Note that P_j will know the minimum length of an S-interval sufficient for at least two loops (associated with P_j) to be found in the interval. Initially, this length will depend on the stage at which P_j steps into the construction. Later, if P_j responds, this length can be dropped to a constant dependent on j (more specifically, 2^{j+2}). The outcomes in the general case are similar, and the strategies still act independently. See [KMN17b] for a further explanation.

2.2.4. *Making the chains finite.* In the subsection above the whole structure was assumed to be one single S-chains with complicated patterns of U-loops, and with another unary function r which maps every point to the single generator – the left-most point in the S-chain.

Clearly, our structure will be much more complicated than just one chain. For that, we should be able to *close* a chain and start a new one.

> *Suppose at a stage of a construction we are monitoring $P_0, \dots, P_k$. If each of these structures either responded by giving the right patterns or have been declared dead, we can* **finish** *the current S-chain by declaring $S(x) = x$ for the right-most point. We say that the chain is now* closed. *We immediately start a new, disjoint S-chain which is built using updated patterns. The patterns are updated according to the basic pressing strategies for $P_0, \dots, P_k$ and the first-attended P_{k+1}.*

Note that making the chains finite does not change the essence of the pressing strategy. We still can recognise the "coordinates" of any point of P_i using the same argument as in the case of a single S-chain. This idea was first used in [KMN17b].

Notation 2.3. *We will also use another unary function p to connect chains. The unary function will be used to map the final element of a chain to the root (the generator) of some other chain.*

We will view every finite S-chain as a substructure. Its isomorphism type will be uniquely determined by the patterns of loops used in its definition. Furthermore, we will guarantee that any isomorphic pair of finite S-chains will be automorphic.

Notation 2.4. *For future convenience, we will use letters with subscripts do denote finite chains. We imagine that the S-chains are build from left to right, with the generator being the left-most point.*

- $\underline{\underline{x_3}}$ *means that x_3 has been finished, and x_3 means that the chain is still being built.*
- $\underline{\underline{a_1}} \leftarrow \underline{\underline{a_2}}$ *means that the final point of the chain a_2 is mapped to the left-most point of the closed chain a_1 under the unary function p. We note that only closed chains can be mapped to chains, and only to closed chains.*
- $\underline{\underline{c_1}} \dashleftarrow c_2$ *means that we intend to map c_2 to $\underline{\underline{c_1}}$ as soon as c_2 is declared closed.*

2.2.5. *Chains of chains, and a special binary relation K.* We can form *chains of chains*, abbreviated *ch.ch.*, using the unary function p. Using the chains of chains we will be able to put more than one finite chain of some fixed isomorphism type into the construction without upsetting the pressing strategies.

For that, we will be using a new binary relational symbol K. We illustrate this procedure in the example below.

Example 2.5. We emphasise that the steps (1)-(5) below do not guarantee the correctness of the procedure. The necessary modification to the steps, which involves K, will be described in detail after we describe the 5 steps below.

(1) x_1.
(2) $\underline{\underline{x_1}}$.
(3) $\underline{\underline{x_1}} \dashleftarrow x_2$.
(4) $\underline{\underline{x_1}} \leftarrow \underline{\underline{x_2}}$, initiate x_3.
(5) $\underline{\underline{x_3}}$, and instantly $\underline{\underline{x_1'}} \leftarrow \underline{\underline{x_3}}$, where $x_1' \cong x_1$ is a new chain.

As the result of these actions we obtain $\underline{\underline{x_1'}} \leftarrow \underline{\underline{x_3}}$ and $\underline{\underline{x_1}} \leftarrow \underline{\underline{x_2}}$,, with $x_1' \cong x_1$. However, the opponent's structure $P = P_e$ could give us its copy of $\underline{\underline{x_1'}}$ too early, long before we finish or even initiate x_3. Another possibility is that P shows x_1' before it shows x_1. But then P will have to show x_2 according to the pressing strategy, and therefore he will have to show us his version of x_1 before we close x_3. Either way, *we will see x_1 and x_1' together in P before we put them into our structure.* In this case:

(a) We ask P to calculate K on the generators of both x_1 and x_1'; we abuse notation and write $K(x_1, x_1')$. (We note that x_1 does not have to be closed here. The trick works at any stage of the procedure described above in (1)-(5).)
(b) As soon as the answer is known, all we need to do is to make the value of K different in our structure when we finally introduce x_1' to our structure $\mathcal{A}$.

It follows that in this scenario $P \ncong \mathcal{A}$.

Not that the same trick with K can be used to prevent the opponent from showing any pair of chains too early (these two do not have to be isomorphic chains). As long as we have not seen these two chains together in our copy, we can later "kill" P by defining K differently from what it is in P on this pair.

The overall conclusion for this section is:

If $P_e \cong \mathcal{A}$ then P_e cannot reveal itself too early.
It must follow $\mathcal{A}$ lagging behind and copying it with a (punctual) delay.

2.3. **One basic strategy in isolation.** We will follow the notation and terminology introduced in the previous section. In particular, we will be forming (finite) chains according to the instructions of the pressing strategy as described in Subsection 2.2.

Initially, start building $\mathcal{A}$ and $\mathcal{B}$ as follows:

- Put a chain x_1 into $\mathcal{A}$ and a chain x_2 into $\mathcal{B}$.
- Do not close (finish) the chain until P_0 chooses to either copy x_1 or to copy x_2; however, still keep the chains open.
- Wait for $p_e : \mathcal{A} \to \mathcal{B}$ to prove that it is not an isomorphism.

Remark 2.6. We pause the description of the strategy to emphasise the implicit use of a binary predicate K which is crucial even at the first stages of the strategy. We must make sure $\mathcal{A} \cong \mathcal{B}$, and therefore at some point in the future we must introduce x_1 to $\mathcal{B}$ and x_2 to $\mathcal{A}$. The opponent's structure may try to reveal both x_1 and x_2 too early (to be more precise, not x_1 and x_2 but their recognisable fragments). But in this case we evaluate K on x_1 and x_2. Note that x_1 and x_2 have not yet been seen together in $\mathcal{A}$ (or $\mathcal{B}$). Later, when we finally put both chains into $\mathcal{A}$ (and $\mathcal{B}$), we will define K differently on them, thus making sure $\mathcal{A} \ncong P$.

We also note that, unless there is a specific instruction for K, we set K equal to 1. We resume the strategy below.

- After the waits above have been finished, close x_1 in $\mathcal{A}$ and x_2 in $\mathcal{B}$.
- Immediately initiate x_3 in $\mathcal{A}$ and x_4 in $\mathcal{B}$.

Remark 2.7. Currently $\mathcal{A}$ consists of $\underline{\underline{x_1}}$ and x_3, and $\mathcal{B}$ contains only $\underline{\underline{x_2}}$ and x_4.

- Wait for P to show a segment of x_3 or prove $P \ncong \mathcal{A}$. (Recall that P follows $\mathcal{A}$.)

Remark 2.8. Note that, according to the description of finite chains given in the previous section, as soon as we recognise a segment l of x_3 we can, with a bounded delay, reconstruct the chain connecting the origin of x_3 with l. Also, if P chooses to show some other pattern which we plan to include into $\mathcal{A}$ later, we use K (as described above) to ensure $P \ncong \mathcal{A}$.

- Once P responds, initiate the $\mathcal{B}$-recovery and $\mathcal{A}$-recovery stages (simultaneously) as described below. Note that x_3 and x_4 have not yet been declared finished yet.
- $\mathcal{A}$-recovery: Using a fresh point on x_3 which has just been introduced and a unary function p (see Notation 2.3), map this point of x_3 to $\underline{\underline{x_2}}$ (which must be instantly introduced in $\mathcal{A}$). This forces P to introduce x_2 too:

  Remark 2.9. Currently $\mathcal{A}$ consists of $\underline{\underline{x_1}}$ and $\underline{\underline{x_2}} \leftarrow x_3$.

- $\mathcal{B}$-recovery: Simultaneously with $\mathcal{A}$-recovery, use a fresh point along the x_4-chain and p to put $\underline{\underline{x_1}}$ into $\mathcal{B}$.

  Remark 2.10. Currently $\mathcal{B}$ consists of $\underline{\underline{x_1}} \leftarrow x_4$ and $\underline{\underline{x_2}}$. Of course, in isolation we would not have to be too careful with $\mathcal{B}$ because P has chosen to copy $\mathcal{A}$. However, in general care must be taken, so we treat $\mathcal{A}$ and $\mathcal{B}$ symmetrically.

- Close x_3 and x_4.

After the module above has finished its work, immediately restart the strategy using fresh chains x_5 and x_6 in $\mathcal{A}$ and $\mathcal{B}$ instead of x_3 and x_4, respectively. We diagonalise against another potential isomorphism, this time from $\mathcal{B}$ to $\mathcal{A}$. Later, use fresh points on these chains and p to put $\underline{\underline{x_4}}$ into $\mathcal{A}$ and $\underline{\underline{x_3}}$ into $\mathcal{B}$. Then repeat this with x_7 and x_8 to diagonalise against the next potential isomorphism from $\mathcal{A}$ to $\mathcal{B}$, etc. Note that in the limit $\mathcal{A} \cong \mathcal{B}$.

Remark 2.11. If we ignore the exact definition of K which will depend on the construction, then the isomorphism type of both $\mathcal{A}$ and $\mathcal{B}$ can be sketched as follows:

$$\begin{array}{c}
\underline{\underline{x_1}} \leftarrow \underline{\underline{x_4}} \leftarrow \underline{\underline{x_5}} \leftarrow \underline{\underline{x_8}} \leftarrow \ldots \leftarrow \underline{\underline{x_i}} \leftarrow \underline{\underline{\underline{x_{i+3}}}} \leftarrow \underline{\underline{\underline{x_{i+4}}}} \leftarrow \ldots\ldots \\
\underline{\underline{x_2}} \leftarrow \underline{\underline{x_3}} \leftarrow \underline{\underline{x_6}} \leftarrow \underline{\underline{x_7}} \leftarrow \ldots \leftarrow \underline{\underline{\underline{x_{i+1}}}} \leftarrow \underline{\underline{\underline{x_{i+2}}}} \leftarrow \underline{\underline{\underline{x_{i+5}}}} \leftarrow \ldots.
\end{array}$$

Obviously, the isomorphism type of each chain will also depend on the construction.

2.4. **The case of two structures P_0 and P_1.** Initially, we monitor only P_0. The exact stage at which we finally start considering P_1 depends on us. This delay will not effect the construction because it does not delay the enumerations of $\mathcal{A}$ and $\mathcal{B}$. In particular, before we start considering P_1 we wait for P_0 to start copying either $\mathcal{A}$ or $\mathcal{B}$ or be "killed" using K.

We make sure that when P_1 finally steps into then construction P_0 has already made its choice. If $\mathcal{A} \ncong P_0$ then the analysis of P_1 is identical to that in the previous subsection. Thus, without loss of generality, assume that P_0 is currently copying $\mathcal{A}$.

- Suppose x_i is open in $\mathcal{A}$ and x_j in $\mathcal{B}$.
- Wait for P_1 to either reveal a segment of x_i or a segment of x_j.
- If in the process P_1 reveals some of its parts too early (see the previous subsection) then declare P_1 *ready for execution.*

While we monitor P_1 we also keep observing P_0 because we have to punctually define an isomorphism between P_0 and $\mathcal{A}$. If P_0 reveals itself too quickly, it must also be immediately declared ready for execution.

There are three cases to consider.

2.4.1. *Case 1: P_1 has been declared ready for execution.* If P_0 keeps obediently following $\mathcal{A}$, then the next time we introduce the missing chains into $\mathcal{A}$ and $\mathcal{B}$ we will use K to ensure $P_1 \ncong \mathcal{A}$ in the same way we did it in the previous section. However, it could be the case that P_0 also reveals its parts too early and thus is declared ready for execution.

The obvious conflict there is that the value of K on a pair of points (which we intend to use for diagonalization) may be different in P_0 and P_1. Say, we are planning to use the left-most points a and

b of x_i and x_k (resp.) in both P_0 and P_1, but $K_{P_0}(a,b) \neq K_{P_1}(a,b)$. The fix however is trivial. Simply use the successors of a and b along the respective chains x_i and x_k to diagonalise against P_1, and use a and b to "kill" P_0.

More generally, *we reserve the k'th point of each chain as a potential witness for diagonalization against P_k.* This way there will be no interaction between the diagonalization strategies because they will use K on distinct points.

2.4.2. *Case 2: Both P_1 and P_2 copy $\mathcal{A}$.* This is similar to the description of one P_0 in isolation, but now we have to wait for both P_0 and P_1 to respond in the basic pressing strategy according to its description in Subsection 2.2; this process has already been described in Subsection 2.2. Any action for the sake of P_1 has to be delayed until P_0 gives more evidence that $\mathcal{A} \cong P_0$. In particular, if P_1 is declared ready for execution, we *first* finish all actions associated with P_0 and *then* we can diagonalise against P_1 using K and the reserved witnesses in the respective chains.

2.4.3. *Case 3: P_1 copies $\mathcal{A}$ and P_2 copies $\mathcal{B}$.* This is essentially the same as Case 2 above, but simpler because the basic pressing technique is now essentially acting independently in the currently active chains in $\mathcal{A}$ and $\mathcal{B}$ (because they are pressing different structures). As in Case 2, we always wait for P_0 to respond before taking any action for the sake of P_1. The diagonalization with the help of K is also similar. Since we agreed to use different points of the chains as witnesses for K, there is essentially no interaction or conflict between the two diagonalization strategies working with P_0 and P_1, respectively.

2.5. **The construction. The general case of all P_e.** In the construction, we will slowly increase the number of monitored structures P_e. At every stage we monitor only finitely many of them. Only after each of them has responded again or has been diagonalised against, will we start looking at the next structure in the list. The formation of the simple chains x_i in presence of many P_e has already been described in Subsection 2.2. Since P_i and P_j will use different witnesses for K, there is no conflict between the diagonalization K-strategies working with different structures. Thus, in the construction we let all the strategies act according to their instructions as described above; no further modifications are necessary.

2.6. **Verification.** Much of the verification was incorporated explicitly into the description of the strategies. For instance, it is clear that $\mathcal{A}$ and $\mathcal{B}$ are lagebraically isomorphic, but they cannot be punctually isomorphic because we have diagonalised against each potential punctual isomorphism from $\mathcal{A}$ to $\mathcal{B}$ and from $\mathcal{B}$ to $\mathcal{A}$.

It takes a bit more effort to show that each $P_e \cong \mathcal{A}$ either is punctually isomorphic to $\mathcal{A}$ or is punctually isomorphic to $\mathcal{B}$. We split it into several claims.

Claim 2.12. *If P_e initially chooses to copy $\mathcal{A}$ (or $\mathcal{B}$) then it will either be diagonalised against or will forever keep copying $\mathcal{A}$ (resp., $\mathcal{B}$).*

Proof. Assume P_e has initially chose to follow $\mathcal{A}$. If P_e ever attempts to not follow $\mathcal{A}$ by revealing some pattern so far unseen, the basic pressing strategy (Subsection 2.2) will ensure $P_e \ncong A$ by forbidding this pattern from use in the construction. If $\mathcal{A}$ attempts to show a part of $\mathcal{B}$ not yet enumerated into $\mathcal{A}$, then it will be declared ready for execution and will be diagonalised against (and in finite time) using the special binary predicate K, as described above. □

Note that $\mathcal{A} \cong \mathcal{B}$ is rigid and consists of two (infinite) chains of (finite) chains.

Claim 2.13. *If P_e initially chooses to copy $\mathcal{A}$ (or $\mathcal{B}$) then P_e and $\mathcal{A}$ are punctually isomorphic*[1].

Proof. The description of the pressing strategy allows us to for a set of local "coordinates". As described in Subsection 2.2, using these "coordinates" we can punctually map points in P_e to points in $\mathcal{A}$ if P_e initially chose to copy $\mathcal{A}$. Punctually mapping points in $\mathcal{A}$ to points in P_e requires a bit more care. The pressing strategy in Subsection 2.2 does not take into account the following scenario. It could be the case that P_e initially chose to copy $\mathcal{B}$ by giving a pattern in the chain x_j which is currently being

[1]Meaning that both the isomorphism and its inverse are primitive recursive.

built in $\mathcal{B}$ but is not yet present in $\mathcal{A}$. Then x_j will be eventually mapped to roughly a half of the chains currently present in $\mathcal{B}$, but this delay is not punctual. The other half will be forced to appear in P_e too, due to the actions on a recovery stage; that is, another fresh chain will eventually be put into $\mathcal{B}$, then much later closed and mapped to the "other half" of $\mathcal{B}$ via p. This delay is also not punctual. However, we can use the stage at which these processes finally happen *as a non-uniform parameter*. After all chains currently present in $\mathcal{B}$ are forced to appear in P_e, we can use the "coordinate system" defined by the pressing strategy to punctually map any point in $\mathcal{B}$ to a point in P_e. □

The verification is finished, and the theorem is proved.

3. Conclusion

Recall that the inverse of a primitive recursive function does not have to be primitive recursive. This leads us to a natural *reduction* on the set of all punctual copies of a structure:

Definition 3.1. Let $\mathcal{A}$ be a punctual structure. Then, for punctual $\mathcal{C}, \mathcal{B}$ isomorphic to $\mathcal{A}$,

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

The associated equivalence relation $\cong_{pr}$ and the reduction between its classes gives *the punctual degree structure* of $\mathcal{A}$ which will be denoted $PR(\mathcal{A})$. Immediately, we run into a surprisingly challenging question:

Question 3.2. *Is there a structure $\mathcal{A}$ such that $PR(\mathcal{A})$ consists of exactly one degree yet $\mathcal{A}$ is not punctually categorical?*

Note that the former means that every pair of copies have primitive recursive isomorphisms in both directions, but this does not formally imply the existence of a punctual f between them (with both f and f^{-1} primitive recursive). it is not hard to see that if a structure is finitely generated then $|PR(\mathcal{A})| = 1$ is the same as being punctually categorical. Melnikov and Ng [MN] used a rather involved argument to prove that the same holds for graphs. It is not even clear at present if their proof can be extended to cover ternary relational structures or unary functional structures. As it stands, the question above is open.

What does $PR(\mathcal{A})$ reflect? If $\mathcal{C} \leqslant_{pr} \mathcal{B}$ then, in a way, $\mathcal{B}$ has more online content than $\mathcal{C}$ does. For example, the standard copy of $(\mathbb{Q}, <)$ punctually embeds any other punctual copy of the rationals, but some other copies may have slow intervals. The FPR degrees serves as a punctual invariant of a structure. A reader familiar with the terminology of computable structure should compare FPR degrees with degree spectra and categoricity spectra of structures. Note that non-trivial degree or categoricity spectra are typically realised by unnatural structures which have to be specifically constructed; see, e.g., [Mil01, KKM13, FKH+12, FKM10]. In contrast, the FPR degrees of natural and common algebraic structures such as the dense linear order or the random graph seem to possess remarkably non-trivial properties which are yet to be understood. Therefore, it makes sense to study the FPR degrees of specific and natural algebraic structures and compare them with FPR degrees of other structures. For example, the FPR degrees of the dense linear order, the random graph, and the universal countable abelian p-group are pairwise non-isomorphic; see [MN]. It is not known whether the FPR degrees of $(Q, <)$ and the atomless Boolean algebra are isomorphic [BDKM].

Our proof above of Theorem 1.3 shows that there exists a structure $\mathcal{A}$ whose punctual degrees $PR(\mathcal{A})$ consist of exactly two incomparable degrees. We leave open:

Question 3.3. *Is there a structure $\mathcal{A}$ such that $PR(\mathcal{A})$ consists of finitely many (and more than one) degrees and so that not all of them are incomparable?*

In fact, we do not even know the answer to the seemingly less challenging (but closely technically related) question below.

Question 3.4. *Is there a structure $\mathcal{A}$ such that $PR(\mathcal{A})$ contains more than one degree and so that $PR(\mathcal{A})$ is linearly ordered under $\leqslant_{pr}$?*

We suspect that new insights will be needed to make progress in these two questions. We also believe that answering one of the two questions will likely give a way to answer the other one.

References

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

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

[Ala18] P. E. Alaev. Structures computable in polynomial time. II. *Algebra Logic*, 56(6):429–442, 2018.

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

[Bor09] É. Borel. Les probabilités dénombrables et leurs applications arithmétiques. *Rend. Circ. Mat. Palermo*, 27(1):247–271, 1909.

[CDRU09] Douglas Cenzer, Rodney G. Downey, Jeffrey B. Remmel, and Zia Uddin. Space complexity of abelian groups. *Arch. Math. Log.*, 48(1):115–140, 2009.

[CR] D. Cenzer and J.B. Remmel. Polynomial time versus computable boolean algebras. *Recursion Theory and Complexity, Proceedings 1997 Kazan Workshop (M. Arslanov and S. Lempp eds.), de Gruyter (1999), 15-53.*

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

[CR92] Douglas A. Cenzer and Jeffrey B. Remmel. Polynomial-time abelian groups. *Ann. Pure Appl. Logic*, 56(1-3):313–363, 1992.

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

[CS19] Barbara F. Csima and Jonathan Stephenson. Finite computable dimension and degrees of categoricity. *Ann. Pure Appl. Logic*, 170(1):58–94, 2019.

[Deh11] M. Dehn. Über unendliche diskontinuierliche Gruppen. *Math. Ann.*, 71(1):116–144, 1911.

[ECH$^+$92] David B. A. Epstein, James W. Cannon, Derek F. Holt, Silvio V. F. Levy, Michael S. Paterson, and William P. Thurston. *Word processing in groups*. Jones and Bartlett Publishers, Boston, MA, 1992.

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

[FHM14] Ekaterina B. Fokina, Valentina Harizanov, and Alexander Melnikov. Computable model theory. In *Turing's legacy: developments from Turing's ideas in logic*, volume 42 of *Lect. Notes Log.*, pages 124–194. Assoc. Symbol. Logic, La Jolla, CA, 2014.

[FKH$^+$12] A. Frolov, I. Kalimullin, V. Harizanov, O. Kudinov, and R. Miller. Spectra of high$_n$ and non-low$_n$ degrees. *J. Logic Comput.*, 22(4):755–777, 2012.

[FKM10] Ekaterina B. Fokina, Iskander Sh. Kalimullin, and Russell Miller. Degrees of categoricity of computable structures. *Arch. Math. Log.*, 49(1):51–67, 2010.

[Gon80] S. Goncharov. The problem of the number of nonautoequivalent constructivizations. *Algebra i Logika*, 19(6):621–639, 745, 1980.

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

[Gri90] Serge Grigorieff. Every recursive linear ordering has a copy in $DTIME\text{-}SPACE(n, log(n))$. *J. Symb. Log.*, 55(1):260–276, 1990.

[Her26] Grete Hermann. Die Frage der endlich vielen Schritte in der Theorie der Polynomideale. *Math. Ann.*, 95(1):736–788, 1926.

[Hir99] Denis Roman Hirschfeldt. *Degree spectra of relations on computable structures*. ProQuest LLC, Ann Arbor, MI, 1999. Thesis (Ph.D.)–Cornell University.

[HKSS02] D. Hirschfeldt, B. Khoussainov, R. Shore, and A. Slinko. Degree spectra and computable dimensions in algebraic structures. *Ann. Pure Appl. Logic*, 115(1-3):71–113, 2002.

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

[Kie98] H. A. Kierstead. On line coloring k-colorable graphs. *Israel J. Math.*, 105(1):93–104, 1998.

[KKM13] I. Kalimullin, B. Khoussainov, and A. Melnikov. Limitwise monotonic sequences and degree spectra of structures. *Proc. Amer. Math. Soc.*, 141(9):3275–3289, 2013.

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

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

[KN94] Bakhadyr Khoussainov and 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*, pages 367–392, 1994.

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

[KS99] Bakhadyr Khoussainov and Richard A. Shore. Effective model theory: the number of models and their complexity. In *Models and computability (Leeds, 1997)*, volume 259 of *London Math. Soc. Lecture Note Ser.*, pages 193–239. Cambridge Univ. Press, Cambridge, 1999.

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

[Mal61] A. Mal′cev. Constructive algebras. I. *Uspehi Mat. Nauk*, 16(3 (99)):3–60, 1961.

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

[Mil01] R. Miller. The Δ^0_2-spectrum of a linear order. *J. Symbolic Logic*, 66(2):470–486, 2001.

[Mil11] Russell Miller. An introduction to computable model theory on groups and fields. *Groups Complexity Cryptology*, 3(1):25–45, 2011.

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

[MN82] G. Metakides and A. Nerode. The introduction of nonrecursive methods into mathematics. In *The L. E. J. Brouwer Centenary Symposium (Noordwijkerhout, 1981)*, volume 110 of *Stud. Logic Found. Math.*, pages 319–335. North-Holland, Amsterdam, 1982.

[MPSS18] Russell Miller, Bjorn Poonen, Hans Schoutens, and Alexandra Shlapentokh. A computable functor from graphs to fields. *J. Symb. Log.*, 83(1):326–348, 2018.

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

[Tsa11] Todor Tsankov. The additive group of the rationals does not have an automatic presentation. *J. Symbolic Logic*, 76(4):1341–1351, 2011.

[vM19] R. von Mises. Grundlagen der Wahrscheinlichkeitsrechnung. *Math. Z.*, 5(1–2):52–99, 1919.

Massey University Auckland, Private Bag 102904, North Shore, Auckland 0745, New Zealand
Email address: `alexander.g.melnikov@gmail.com`

NanyanG Technological University, Singapore