PROCEEDINGS OF THE
AMERICAN MATHEMATICAL SOCIETY
Volume 00, Number 0, Pages 000–000
S 0002-9939(XX)0000-0

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 ω 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.

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 uniting these independent subjects. 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 dimension of 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 above-mentioned proof in [Gon80] or with any other known computable dimension two proof.

1.1. **Turing computable mathematics.** The general area of *computable* or *effective mathematics* is devoted to understanding the algorithmic content of mathematics. The standard model for such investigations is a (Turing) *computable presentation* of a countable structure. By this we mean a presentation of the structure with universe $\mathbb{N}$, and the relations and functions coded as Turing computable objects. 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].

The first author is partially supported by the Marsden Foundation of New Zealand. The second author is partially supported by the grants MOE2015-T2-2-055 and RG131/17.

One popular topic in such investigations has been 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 computable presentation up to 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 dichotomy. Remarkably, via an intricate argument Goncharov [Gon80] constructed an algebraic structure of *computable dimension two*; that is, a structure which has *exactly two* computable presentations up to computable isomorphism. Although the first such structure was algebraically artificial, similar examples were later found among two step nilpotent groups [Gon81], (remarkably) fields [MPSS18], and some other natural classes [HKSS02]. There has been many further 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 process. 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 further restrict the notion of "computability" by putting 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. Gregorieff, Cenzer and others [CR98, Gri90] and more recently Alaev and Selivanov [Ala17, Ala18, AS18] studied the much more general notion *polynomial time* presentable structures. We omit the formal definitions, but we note that they are again sensitive to how exactly we represent the domain of a structure. 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, Melnikov 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. Furthermore, to illustrate that a structure has no polynomial time copy, it is typically easiest to argue that it does not even have a copy with primitive recursive operations; see, e.g., [CR92]. 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 be total (i.e., simply eventually halts), and there is *no resource bound* imposed otherwise.

Recall that 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. 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 unspecified delay", where these notions are formalised in the sense that we can precompute the bound. Irrelevant counting combinatorics is stripped off such 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 *punctual*[1] if its domain is $\mathbb{N}$ and the operations and predicates of the structure are (uniformly) primitive recursive.

The intuition is that a punctual structure must reveal itself "punctually", i.e., within a precomputed number of steps. We will also fix the convention 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 need to clarify what uniformity means in Definition 1.1.

To talk about isomorphisms and categoricity in this framework, we shall also need to consider "punctual" analogues of computable functions. However, recall that the inverse of a primitive recursive function does not have to be primitive recursive. This gives rise to, for instance, several possible ways of defining when two punctual structures are "punctually isomorphic". We shall only consider the following strongest such notion:

Definition 1.2 ([KMN17b]). A function $f : \mathbb{N} \to \mathbb{N}$ is *punctual* if both f and f^{-1} are primitive recursive. A structure is *punctually categorical* if it has a unique punctual presentation up to punctual isomorphism.

[1]In [KMN17b], the authors used the term "fully primitive recursive".

Punctual isomorphisms appear to be the most natural morphisms in the category of punctual structures. If $\mathcal{A}$ and $\mathcal{B}$ are punctual structures, we write $\mathcal{A} \cong \mathcal{B}$ to mean that they are isomorphic, and $\mathcal{A} \cong_{pr} \mathcal{B}$ to mean that there is a punctual isomorphism from $\mathcal{A}$ onto $\mathcal{B}$. The definition above ensures that $\cong_{pr}$ is an equivalence relation.

We mention a related notion. If $\mathcal{A}$ and $\mathcal{B}$ are punctual copies of some countable structure, we say that $\mathcal{A} \leqslant_{pr} \mathcal{B}$ if there is a primitive recursive isomorphism from $\mathcal{A}$ onto $\mathcal{B}$. This is clearly a preordering and the induced equivalence relation is denoted by $\equiv_{pr}$. Obviously, $\mathcal{A} \cong_{pr} \mathcal{B}$ implies that $\mathcal{A} \equiv_{pr} \mathcal{B}$, and it is open (see Question 3.1) whether having a $\equiv_{pr}$-equivalence class of size 1 is equivalent to being punctually categorical in general.

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 considered in [KMN17b] a structure has either one or infinitely many punctual copies up to punctual isomorphism. 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 had 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 former does not a priori imply the latter; nonetheless, all naturally occurring examples strongly suggested that the implication should hold. After several years of investigation, we have finally accumulated enough intuition and technical tools to refute the "1 vs. ω" conjecture for punctual structures.

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

The theorem solves a problem left open in [Mel17]; see also [BDKM]. The proof combines a "pressing" strategy from [KMN17b] with a new technique. We emphasise that our proof shares virtually nothing in common with Goncharov's dimension two proof. The only similarity is perhaps their relatively high combinatorial complexity and the use of some patterns to "press" the opponent. We believe that both the result and the new technique introduced in its proof will find important applications in the theory of punctual structures, and perhaps beyond. In fact, we will shortly mention one such recent application.

We strongly believe that our proof can be modified to produce a structure with exactly n punctually incomparable copies, for each $n \in \omega$. We leave this as a conjecture. We also strongly suspect that the construction actually produces a polynomial time structure which has dimension two with respect to polynomial time isomorphisms, but recall that the notion of "polynomial time" depends on how exactly one represents the domain; see [CR98]. In particular, we conjecture that under the unary representation of $\mathbb{N}$ (i.e, n is identified with the string of zeros of length n) our construction as is already gives a structure of polynomial-time dimension two. One can perhaps adjust our proof to produce an example of this sort for binary polynomial-time representations. We leave the investigation of the polynomial-time case as an open problem.

Finally, we would like to know whether algebraically natural classes, such as fields or groups, contain examples of finite punctual dimension. Here the situation is a lot more complex that in the Turing computable case, because many of the (Turing) universal classes turned out to be not punctually universal; see [BDKM, DHTK+18, HTMMM17] for definitions. For instance, in the Turing computable world structures with only two unary functions are computably universal. Downey, Greenberg, Melnikov, Ng and Turetsky have recently announced that unary structures are not punctually universal. Their complex proof relies on a novel strategy introduced in the proof of Theorem 1.3.

The rest of the paper is devoted to the proof of Theorem 1.3. We also state a related open problem 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 eth punctual presentation in a fixed total computable enumeration of all punctual structures of the language (recall that the domain of each P_e has to be the whole of ω). Note that the enumeration $P_0, P_1, \cdots$ can be done in a computable, but not in a primitive recursive way. Namely, there is a total computable, but no primitive recursive function Q such that $Q(e,k) = P_e(k)$.

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 eth primitive recursive function in a fixed total enumeration of all primitive recursive functions. Again, this listing $p_0, p_1, \cdots$ is a computable, but not primitive recursive listing. We of course only need to show $\mathcal{A} \not\cong_{pr} \mathcal{B}$ and hence only need to worry about those p_e which are punctual. The requirements p_e ensure that $\mathcal{A}$ and $\mathcal{B}$ are in fact $\leqslant_{pr}$-incomparable. Apart from being easier to implement, the stronger requirements will allow us to say something about $\equiv_{pr}$ degree structures:

Corollary 2.1. *There is a punctual structure $\mathcal{A}$ with exactly two $\equiv_{pr}$-degrees (amongst all punctual copies of $\mathcal{A}$), and the two degrees are incomparable.*

This conclusion may be interesting to the reader who wishes to study the structure of the degrees arising by considering the preordering $\leqslant_{pr}$ (see [MN]).

2.2. **The pressing strategy.** In this section we describe the key strategy for proving Theorem 1.3, which we call the *pressing strategy.* To illustrate the pressing strategy in the most basic form, we formulate simplified versions of our requirements in this section for the purpose of this discussion.

We consider only $\mathcal{A}$ and attempt to meet, for each e, the requirement

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

where $(P_e)_{e\in\omega}$ is the natural uniformly computable listing of all punctual structures. This requirement is known as "pressing P_e", as the requirement will ensure that if P_e is a copy of $\mathcal{A}$ then in order to have $P_e \cong_{pr} \mathcal{A}$, we will need to build $\mathcal{A}$ in a particular way in order to force certain local patterns to be generated quickly in P_e. The reader should think of P_e as of being "increasingly slow" as e increases. 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 convergence 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 node $S^n(0)$. To be more specific, suppose we attach 2-loops. Use another unary function r that sends each point (in the U-loops as well as in the S-chain) 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 obviously 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 $P_0[s]$, so that if the U-chain eventually closes into a loop, the resulting size of the U-loop will be larger than anything currently used in $\mathcal{A}$, and we can then kill P_0 by forever forbidding this size, similar to (2) above.

Notice that after iteratively applying the operations of P_0 on any element of P_0 at most three times, we will be able to tell if (1), (2) or (3) above holds. If one of these three cases hold, we can switch to satisfying P_0 by forcing $P_0 \not\cong \mathcal{A}$. Therefore, assume none of the above cases apply. This means that P_0 responds by giving us a few consequent 2-loops. Note that in order for us not to be able to kill P_0 as above, we must see a U-loop of size 2 after three iterations of P_0 operations. Notice that this is primitive recursive relative to the structure P_0, in the sense that this process can be time-bounded by a primitive recursive transformation of the operations mentioned in P_0.

The reader who is new to this strategy may now wonder why we need U-loops in our structure $\mathcal{A}$; after all, is the "root pointer function $r(x)$" not enough to carry out the above pressing strategy? The answer lies in the observation that as $\mathcal{A}$ is rigid, in order to have $\mathcal{A} \cong_{pr} P_0$, we need to not only map the root of A to the root of P_0, but must also preserve the distance of elements to the root. Having $r(x)$ merely forces P_0 to generate 0^{P_0} quickly, but P_0 could, for instance introduce an element $x \in P_0$ and keep the distance of x to the root undeclared. More specifically, P_0 may never contain a sequence of elements $0, S(0), \cdots S^n(0)$ such that $S^n(0) = x$. Since we have to declare the preimage of x in $\mathcal{A}$ relatively quickly, as we have to ensure that $\mathcal{A} \cong_{pr} P_0$, the structure might only show such a sequence (and hence reveal the true distance of x from 0^{P_0}) only after we have declared the preimage of x, and cause our attempt at showing $\mathcal{A} \cong_{pr} P_0$ to be wrong. Note that even including the "predecessor function" $x \mapsto S^{-1}(x)$ into the language is insufficient, and we are thus forced to innovate by the use of U-loops.

Our solution is to use different U-loop sizes to press P_0. As soon as P_0 responds by giving two distinct 2-loops in its structure, the first of which is attached to some element x in the primary S-chain of P_0, we will 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 of $\mathcal{A}$. What this means is that from this point on, we will attach U-loops of alternating $2,4$-sizes to subsequent elements in the primary S-chain.

How do we punctually map the element $x \in P_0$ (see above) to $\mathcal{A}$? Equivalently, how can we quickly compute the distance of x from the root in P_0? Recall that $x \in P_0$ had the property that x and $S(x)$ had 2-loops attached to them. In $\mathcal{A}$, the initial segment consisting of adjacent 2-loops has a specific length that we know at the stage, say k, where k is the stage where P_0 has revealed the elements $x, S(x)$ and all of their attached 2-loops. Since after stage k, we switched from the $2-2$ pattern to the $2-4$ pattern, our structure $\mathcal{A}$ has the property that for every element $S^m(0)$ with $m > k$, the loops attached to $S^m(0)$ and $S^{m+1}(0)$ cannot both have size 2. In order for P_0 to be isomorphic to $\mathcal{A}$, the element x in P_0 must have distance at most k to the root. To compute the exact distance of x, we simply evaluate the function S to 0^{P_0} iteratively at most k times, and we must obtain x by then. This process can be time-bounded by a primitive recursive transformation of the operations in P_0, and hence can be repeated to produce a punctual isomorphism between $\mathcal{A}$ and P_0.

To formally 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 the enumeration of P_0 (see the discussion above), and iteratively generate images for each element of $\mathbb{N}$.

2.2.2. *Pressing P_0 and P_1.* For simplicity, the highest priority structure can be pressed using loops attached to the 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 in the strategy for pressing P_0 is that the loops corresponding to P_0 are now located at even positions, rather than at every position:

$$2 - \square - 2 - \square - 2 - \square - 2 - \cdots,$$

where the content of $\square$ does not worry the strategy for P_0. This strategy then switches to:

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

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

$$\cdots - 2 - \square - 4 - \square - 4 - \square - 2 - \square - 4 - \square - 4 \cdots.$$

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

$$\cdots - 2 - \square - 8 - \square - 2 - \square - 8 - \square - 2 - \square - 8 \cdots.$$

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 prime numbers. Then we could slowly introduce longer loops into the construction whenever we are ready to do so. This will allow us to keep the strategies fairly independent from each other.

Remark 2.2. *In several related constructions that uses a similar pressing strategy, we could get around with using* only *loops of size* 2 *and* 4 *for a single pressing strategy. At a late enough stage we will know the exact* 2, 4 *pattern that we have to check in* P_0 *to understand where the respective location is.*

From the perspective of the P_0 pressing strategy, the following scenarios are possible:

- P_0 has an obviously wrong isomorphism type. This is an instant win which requires no further action.
- P_0 shows an S-pattern of size 4 with a loop of size 2^k, where k has not yet been used. Then the strategy forever forbids this pattern, and thus guarantees that P_0 is not isomorphic to our structure.
- P_0 shows an S-pattern of size 4, and at least one of the attached loops could potentially have length of the form 2^k and 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, since our structure will never have a loop of size 2^k for any k larger than all the ones used thus far.

Remark 2.3. Note that, in the third clause we do not worry about the $\square$ components, as long as the $\square$ components use loops of size p^j for $p \neq 2$, and this trick removes some tensions in the construction. We elaborate on it using an example. Suppose we see a sequence $x - y - z - w$ of S-successors. We start evaluating the unary loop function for all of them. Suppose the P_0-strategy had used loops of sizes 2, 4 and 8 so far. 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.

- All the four attached loops have size at most 8, and that $x - y - z - w$ cannot possibly give us a right pattern of powers of 2. This can be decided based on the sizes of the 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 $\Box$ 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 2.4. *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 (of $\mathcal{A}$ or $\mathcal{B}$). 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 2.5. 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 declared closed, 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, and that both chains have been declared finished (or closed). 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. *A special binary relation K.* In the construction, we will be building two punctual copies $\mathcal{A}$ and $\mathcal{B}$ of the same structure. At the end of the construction our structure will consist of infinitely many finite chains, all having distinct isomorphism types. At every stage of the construction $\mathcal{A}$ and $\mathcal{B}$ will consist of different configurations of finite chains interlinked by a unary function. For example, for a number of stages $\mathcal{A}$ will contain a finite chain x_j but $\mathcal{B}$ will not, while $\mathcal{B}$ may contain some other chain x_k which will not be present in $\mathcal{A}$ for a large number of stages.

Recall also that, for each punctual P, we must build a punctual isomorphism either from P to $\mathcal{A}$ or from P to $\mathcal{B}$. The obvious potential conflict here is that P may reveal some parts of *both* x_j and x_k too early, long before $\mathcal{A}$ or $\mathcal{B}$ are ready to accommodate both chains simultaneously.

To resolve this conflict we will be using a new binary relational symbol K, as follows.

In P, evaluate K on the generators of x_j and x_k. Later, when we are ready to put both x_j and x_k into $\mathcal{A}$ (and $\mathcal{B}$), evaluate K differently on the respective pair.

In presence of only one P the idea above clearly prevents P from revealing itself too early. In the general case the idea will have to be slightly modified and blended with the standard priority technique.

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 $\mathcal{B}$.
- Wait for P_0 to either copy x_1 or to copy x_2.
- Wait for $p_e : \mathcal{A} \to \mathcal{B}$ to halt and, thus, 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 $P = P_0$ 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 ask P evaluate K on x_1 and x_2, say, on their left-most generators. 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, 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 either 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 origin of x_3 using 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.

- $\mathcal{A}$-recovery: Using a fresh point along x_3 and a unary function p (see Notation 2.4), 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:

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

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 the construction, P_0 has already made its choice. If we ensure that $\mathcal{A} \ncong P_0$ then P_0 no longer has any effect in the construction, and thus 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[2] 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 evaluate K at 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] That is, if P_1 shows some recognisable parts of chains which are not currently and simultaneously present in either $\mathcal{A}$ or $\mathcal{B}$; see the previous subsection.

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.** 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.** Although a strategy monitoring P_e may have an infinitary outcome (in this case $P_e \ncong \mathcal{A}$), no tree of strategies was necessary. Also, injury in the construction is merely finite. Therefore the combinatorics related to priority is rather tame and, more importantly, standard.

As for the combinatorics specific to primitive recursion, much of it was explained and verified explicitly into the description of the strategies. For instance, it is clear that $\mathcal{A}$ and $\mathcal{B}$ are algebraically 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*[3].

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

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

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 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 loop patterns defined by the pressing strategy to punctually map any point in $\mathcal{B}$ to a point in P_e. $\square$

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.

Question 3.1. *Suppose for any pair $\mathcal{A}$ and $\mathcal{B}$ of punctual presentations of a structure, there exist primitive recursive isomorphisms from $\mathcal{A}$ onto $\mathcal{B}$ and from $\mathcal{B}$ onto $\mathcal{A}$. Does the structure have to be punctually categorical?*

In other words, is there an isomorphism $f : \mathcal{A} \to \mathcal{B}$ with both f and f^{-1} primitive recursive. Note that $\mathcal{A}$ and $\mathcal{B}$ must be arbitrary punctual presentations of the structure. It is not hard to see that if the structure is finitely generated then the answer is positive. Melnikov and Ng [MN] have 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, several binary relations, or unary functional structures.

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.

[AS18] Pavel Alaev and Victor Selivanov. Polynomial-time presentations of algebraic number fields. In *Sailing routes in the world of computation*, volume 10936 of *Lecture Notes in Comput. Sci.*, pages 20–29. Springer, Cham, 2018.

[BDKM] N. Bazhenov, R. Downey, I. Kalimullin, and A. Melnikov. Foundations of online structure theory. Bulletin of Symbolic Logic, 2019.

[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.

[DHTK+18] Rod Downey, Matthew Harrison-Trainor, Iskander Kalimullin, Alexander Melnikov, and Daniel Turetsy. Graphs are not universal for online computability. Preprint, 2018.

[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.

[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 $DTIME$-$SPACE(n, log(n))$. *J. Symb. Log.*, 55(1):260–276, 1990.

[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.

[HTMMM17] Matthew Harrison-Trainor, Alexander Melnikov, Russell Miller, and Antonio Montalbán. Computable functors and effective interpretability. *J. Symb. Log.*, 82(1):77–97, 2017.

[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.

[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.

[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. Israel J.Math, to appear.

[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.

Massey University Auckland, Private Bag 102904, North Shore, Auckland 0745, New Zealand

E-mail address: `alexander.g.melnikov@gmail.com`

Nanyang Technological University, Singapore