GRAPHS ARE NOT UNIVERSAL FOR ONLINE COMPUTABILITY

RODNEY DOWNEY, MATTHEW HARRISON-TRAINOR, ISKANDER KALIMULLIN,
ALEXANDER MELNIKOV, AND DANIEL TURETSKY

Abstract. We show that structures with only one binary function symbol are universal for "online" (punctual) computable structures. In contrast, we give a description of punctually categorical graphs which implies that graphs are *not* universal for online computability.

1. Introduction

There are a large number of natural algorithmic processes which receive input over time (and must pretend that they might receive more input infinitely often) but need to provide output promptly without waiting for more input. For example, consider a memory manager in a computer: over time, the manager receives requests to assign memory, but it cannot wait too long before actually assigning memory. We refer to these as *online* tasks. After decades of development, computability theory and computable structure theory give a well-developed framework to investigate the limits of computation in mathematics. Nonetheless, there is no such general theory for online structures. The paper contributes to the general program [KMN17b, Mel17b, BDKM, KMN17a, MN] that aims to lay the foundations of online computability in algebra and combinatorics. Informally, a computable presentation of a structure is online if more of the structure is computed without delay; we will give formal definitions later.

It is well-known in model theory that graphs are *universal* structures; informally, this means that any structure can be thought of as a graph. This is also true for computable structures. The main result of the paper implies that online presented graphs cannot possess some complex enough features that some other online algebraic structures provably have. That is, online graphs are *not* universal. To state our result formally we need definitions.

We also show that structures with a single binary function are universal for online structures.

1.1. Online structures.

1.1.1. *Computable structures.* The study of general computable processes in algebra is a long tradition going back to the early XXth century; see van der Waarden [vdW30], Hermann [Her26], and Dehn [Deh11]. Maltsev [Mal61] and Rabin [Rab60] proposed the following standard model for such investigations: A *computable presentation* of a structure is a coding of the structure with universe $\mathbb{N}$, so that the relations and functions are Turing computable on the codes of elements. For example, a computable presentation of an infinite group is any 1-1 numbering of its universe with natural numbers such that the group operation becomes a (Turing) computable function on the indices of elements. This is the same as to say that the group has a recursive presentation with solvable word problem [Hig61, LS01].

The general area of computable structures now has a well-developed theory. This area has several threads, as can be seen by the large volumes [EGN$^+$98a, EGN$^+$98b], but below we will discuss a couple of relevance to the present paper.

Turing computability allows to us to use our algorithmic intuition to derive algebraic and definability-theoretic conclusions about a given structure in the general spirit of Hilbert's 10^{th} problem. For example, it is well-known that a relation on a structure is computably enumerable in (relative to) any presentation of a structure $\mathcal{A}$ iff the relation is $\Sigma_1^{\mathcal{A}}$-definable in the language $\mathcal{L}_{\omega_1\omega}$; see the book [AK00]. For explicit connections of such studies with descriptive set theory and model theory see, e.g., Montalbán [Mon13] and Melnikov and Montalbán [MM18]. There has also been a large body of work

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

exploring the relationship between classical isomorphism type and computable isomorphism type. For example, if every pair of computable copies of a structure are computably isomorphic then the structure is called *computably categorical*. Examples of such structures include the dense linear ordering without endpoints and the random graph. It is clear that every pair of computable copies of the random graph are computably isomorphic. We also mention that using more general effective classification tools, such as calibrations via computability-theoretical hierarchies, one can gain understanding of the complexity of classification problems in (not necessarily computable) mathematics [GK02]. For example, we know that certain classes cannot have classical invariants because of the complexity of the isomorphism problem [DM08, Mel17a].

We note that such investigations rely on the most general notion of an algorithm that we know today, namely Turing computable functions. In particular, we do not assume any time or space bounds in such abstract computations. It is a general phenomenon that many such abstract algorithms can be provably turned into more feasible algorithms for reasons yet to be understood; see, e.g., [Gri90, KMN17b]. The present article contributes to the research program which, among other goals, aims to explain this phenomenon. The other important goal is to give a foundation for online structure theory. In particular, we need a representative model of online structures. Following [KMN17b], we choose primitive recursion as the basis of our theory. We will briefly discuss the reasons for this, and then give some precise definitions.

1.1.2. *Resource-constrained computable structure theory.* In the same way that computational complexity theory grew from computability theory, authors have looked at various forms of *feasible* structure theory. That is, 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. For example, treating a function as a relation yields quite a different kind of automatic presentation from treating it is as a transducer. See [ECH$^+$92] for an alternate approach to automatic groups. Although the theory of automatic structures is a beautiful subject, a finite automaton is definitely not general enough for our purposes.

Cenzer and Remmel, Gregorieff, Alaev, and others [CR98, Gri90, Ala17, Ala18] studied *polynomial time* presentable structures. We omit the formal definitions, but we note that they are sensitive to how exactly we code the domain. In many common algebraic classes we can show that all Turing computable structures have polynomial-time computable copies. One attractive result is that every computably presentable linear ordering has a copy in linear time and logarithmic space [Gri90]. Similar results hold for broad subclasses of Boolean algebras [CR] and commutative groups [CR92, CDRU09], and some other structures [CR91]. As was noted in [KMN17b], 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 is sometimes easiest to argue that it does not even have a copy with primitive recursive operations, see e.g. [CR92].

Finally, from a computable model theory perspective, the most natural way to construct structures is via a Henkin construction. A decidable theory has a decidable model, meaning that its full first-order diagram is computable. But 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.

Because of all of the reasons outlined above, we believe that primitive recursive structures provide a general model of the general issues faced in the theory of online structures, in that they give a generic model to focus on the key ingredient, that unbounded search is outlawed. *Our choice for basic online theory is to consider primitive recursive procedures on primitive recursive structures.* It is

natural to systematically investigate those structures that admit a presentation with primitive recursive operations, as defined below

1.1.3. *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*. 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. 1.1.

We have chosen punctual structures as our central model. 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. Furthermore, these models arise quite naturally through standard decision procedures. Irrelevant counting combinatorics is stripped from 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 main result of the paper will be a good illustration of this phenomenon.

1.2. **Graphs are general enough for computable mathematics.** It is well-known that any countable algebraic structure can be algorithmically transformed into a graph. One represents elements of the structure as particular nodes in the graph, each of which is marked with a particular configuration of nodes to distinguish them from the other nodes, and then connects these particular nodes with other configurations to encode the operations and relations. This is simple enough to be considered folklore, but see [RS, Lav63] for the first explicit use of such transformations. Such a coding preserves most of the (Turing) computability-theoretic properties of importance. Classes of countable structures which can effectively "code" any other countable structure are called *universal*. Examples of universal classes also include groups, partial orders, integral domains [HKSS02], and, most notably, fields [MPSS]. In contrast, it follows from [GD80, Rem81, Ric81, LaR77] that Boolean algebras and linear orders are not universal. Harrison-Trainor, Melnikov, Miller, and Montalbán [HTMMM17] have recently shown that all known proofs of universality share the same features. In particular, they are witnessed by Turing functionals that are also functors with some nice additional properties. Furthermore, they showed that there is an equivalent $\mathcal{L}^c_{\omega_1,\omega}$ definability-theoretic formulation of universality in all known cases. Their approach can be taken as the basic formal definition of (Turing) universality of a class; we however omit details.

1.3. **A punctually universal class.** The theorem below gives the first known example of a punctually universal class.

Theorem 1.2. *The class of structures with only one binary function symbol is punctually universal.*

We have not formally defined universality, let alone punctual universality. To state the result formally we need a few elementary definitions. For a total function $f : \omega \to \omega$, let $P(f)$ be the least class containing f and all primitive recursive functions closed under composition and primitive recursion. This is done by forbidding the (unbounded) minimisation operator and adding f to the recursive schemata. Similarly, we can define the notion of a primitive recursive functional.

We must give some definitions in order to state Theorem 1.2 formally. We give a definition of universality in the form of effective categories. This is essentially the same as [MPSS, HTMM15], except that we use p.r. functionals instead of computable functionals. The idea is that a class is universal if we can transform any structure into a structure in that class in a primitive recursive way, and that this tranformation has an inverse transformation.

Definition 1.3. *A* category of structures on ω *is a category where the objects are a set of structures with domain ω, closed under isomorphisms, and the morphisms are the isomorphisms between structures.*

We can view any class of structures closed under isomorphism as a category of structures on ω and vice versa.

Recall that a functor $F\colon \mathcal{C} \to \mathcal{D}$ maps objects in $\mathcal{C}$ to objects in $\mathcal{D}$, and morphisms in $\mathcal{C}$ to morphisms in $\mathcal{D}$, such that:

- for every $\mathcal{A} \in \mathcal{C}$ F must map the identity on $\mathcal{A}$ to the identity on $F(\mathcal{A})$, and
- for all morphisms $g\colon \mathcal{A}_1 \to \mathcal{A}_2$ and $h\colon \mathcal{A}_2 \to \mathcal{A}_3$, $F(h \circ g) = F(h) \circ F(g)$.

A functor is primitive recursive if, essentially, it is given by primitive recursive functionals.

Definition 1.4. *Let $\mathcal{C}$ and $\mathcal{D}$ be categories of structures on ω. A primitive recursive functor is a functor $F : \mathcal{C} \to \mathcal{D}$ for which there exist primitive recursive functionals Φ and Φ^* such that*

- *for every $\mathcal{A} \in \mathcal{C}$, $\Phi(\mathcal{A}) = F(\mathcal{A})$,*
- *for every morphism $g : \mathcal{A} \to \mathcal{B}$, we have $\Phi^*(\mathcal{A} \oplus \mathcal{B} \oplus g) = F(g)$.*

Note that we also get that $F(g^{-1})$ is primitive recursive in g^{-1}.

We also need a notion of two functors being effectively the same. This is somewhat technical.

Definition 1.5. *A functor $F\colon \mathcal{C} \to \mathcal{D}$ is* primitive recursively naturally isomorphic *(or just* p.r. isomorphic*) to a functor $G\colon \mathcal{C} \to \mathcal{D}$ if there are primitive recursive functionals Λ, Λ^{-1} such that for every $\mathcal{A} \in \mathcal{C}$, $\Lambda(\mathcal{A})$ is an isomorphism from $F(\mathcal{A})$ to $G(\mathcal{B})$ and $\Lambda^{-1}(\mathcal{A})$ is its inverse, and the following diagram commutes for every $\mathcal{A}, \mathcal{B} \in \mathcal{C}$ and every morphism $h\colon \mathcal{A} \to \mathcal{B}$:*

$$\begin{array}{ccc} F(\mathcal{A}) & \xrightarrow{\Lambda(\mathcal{A})} & G(\mathcal{A}) \\ {\scriptstyle F(h)}\downarrow & & \downarrow{\scriptstyle G(h)} \\ F(\mathcal{B}) & \xrightarrow[\Lambda(\mathcal{B})]{} & G(\mathcal{B}) \end{array}$$

Two functors are pseudo-inverses if their composition is p.r. isomorphic to the identity.

Now we can define a reduction between categories.

Definition 1.6. *Say that a class $\mathcal{C}$ is* (uniformly) p.r. reducible *to a class $\mathcal{D}$ if there is a subclass $\mathcal{D}'$ of $\mathcal{D}$ and computable functors $F\colon \mathcal{C} \to \mathcal{D}'$, $G\colon \mathcal{D}' \to \mathcal{C}$ such that F and G are pseudo-inverses.*

Note that $\mathcal{D}'$ must be closed under isomorphisms; so the functor G must be defined on any structure isomorphic to one in the image of F. Also note that this is transitive; if $\mathcal{C}$ is p.r. reducible to $\mathcal{D}$, and $\mathcal{D}$ is p.r. reducible to $\mathcal{E}$, then $\mathcal{C}$ is p.r. reducible to $\mathcal{E}$.

Definition 1.7. *A class $\mathcal{D}$ is p.r. universal if for every finite language $\mathcal{L}$, the category of structures in the language $\mathcal{L}$ is p.r. reducible to $\mathcal{D}$.*

Since p.r. functionals are also Turing computable functionals, *every punctually universal class is Turing universal as well.*

It is convenient to use this definition to prove that a class is universal. To show that a class is not universal, it is desirable to give a particular example of a primitive recursive property which cannot be realized within that class; in this way, the fact that the class is not universal in independent of the choice of a precise definition for universality.

We prove Theorem 1.2 in Section 2.

1.4. **Graphs are not general enough for online mathematics.** Recall that a computable structure is *computably categorical* if it has exactly one (Turing) computable presentation up to (Turing) computable isomorphism [EG00, GK02]. We say that a (punctual) structure is punctually categorical if between any two punctual copies of the structure there is a primitive recursive isomorphism with primitive recursive inverse [KMN17b]. In [KMN17b] Kalimullin, Melnikov and Ng constructed an example of a punctually categorical structure which is not computably categorical (and we conjecture that there are punctually categorical structures which are not $0^{(\alpha)}$-categorical). As we noted above, one should expect every punctually universal class to be Turing universal as well. In particular, a punctually universal class must also contain an example of this sort. On the other hand, the following theorem says that punctually categorical graphs are relatively simple.

Theorem 1.8. Let G be an undirected infinite graph. Then the following are equivalent:

(1) G is punctually categorical.
(2) G becomes a clique or an anti-clique (an independent set) after removing finitely many vertices $\bar{v} = v_0, \ldots, v_k$ with each v_i being either adjacent to all $x \in (G - \bar{v})$ or not adjacent to all $x \in (G - \bar{v})$.

In particular, every punctually categorical graph is computably categorical. So graphs are not universal for punctual computability for any reasonable notion of punctual universality.

We remark that the proof of Theorem 1.8 involves several new proof techniques which seem like they should become basic to the area. They are not the usual priority arguments endemic in computability theory, but involve careful understanding of the relative speed of presentations of structures. Also involved is some kind of notion of "distance" within structures, something which is unexplored in classical structure theory in the non-metric setting.

We leave open:

Question 1.9. Is there a punctually universal relational class?

Likely our techniques will make the following sub-question accessible.

Question 1.10. Is there an algebraic description of punctually categorical directed graphs? Of punctually categorical structures with many binary relations? Of punctually categorical structures in a relational language?

2. A punctually universal class: Proof of Theorem 1.2

Throughout the proof we will be often using primitive recursion instead of primitive recursion relative to some total function. In particular, the structures and the isomorphisms are assumed to be primitive recursive. However, it should be rather clear that the argument is relativisable in the right subrecursive sense.

Beginning with the category of structures in a finite language $\mathcal{L}$, we will argue in several steps by giving p.r. reductions to intermediate categories of structures in particular languages. Since p.r. reductions are transitive, this is sufficient. We will also argue somewhat loosely, as it is tedious but straightforward to check for example that two particular functors are pseudo-inverses.

Step 1. Replace all predicate symbols in $\mathcal{A}$ by function symbols as follows (and treat constants as 0-ary functions, so that the language consists entirely of function symbols). Let $\Gamma(\mathcal{A})$ be the extension of $\mathcal{A}$ by two new elements 0, 1 marked by corresponding new constant symbols. Then we can view the predicate symbols from $\mathcal{A}$ as function symbols in $\Gamma(\mathcal{A})$ that map tuples to these constants. It is not hard to see that Γ is a p.r. functor with p.r. pseudo-inverse. Hence, we can assume from the beginning that $\mathcal{A}$ has only function smybols. We also make sure that different function symbols have different arities by replacing, for example, $f(x_1, \ldots, x_n)$ by $f'(x_1, \ldots, x_n, x_{n+1}) = f(x_1, \ldots, x_n)$. Thus without loss of generality we may assume that the signature of $\mathcal{A}$ consists of the functions $f_1, f_2, \ldots, f_m$ and the arity of each f_k, $1 \leq k \leq m$, is equal to k (if some arity k is missing we can introduce a new function $f_k(x_1, \ldots, x_k) = x_1$).

Step 2. Given a functional language with the properties described above, we make the following p.r. reduction. Let $\Gamma(\mathcal{A})$ be the structure with the domain

$$\{[x_1, x_2, \ldots, x_k] : 1 \leq k \leq m \ \& \ x_1, x_2, \ldots, x_k \in \mathcal{A}\}$$

containing all nonempty strings of the length not greater than m in the alphabet $\mathcal{A}$. The structure $\Gamma(\mathcal{A})$ has the binary symbol for truncated concatenation

$$[x_1, x_2, \ldots, x_i] * [y_1, y_2, \ldots, y_j] = [x_1, x_2, \ldots, x_i, y_1, y_2, \ldots, y_{\min(m-i,j)}]$$

and the unary function symbols

$$g_0([x_1, x_2, \ldots, x_k]) = [f_k(x_1, x_2, \ldots, x_k)],$$

$$g_i([x_1, x_2, \ldots, x_k]) = \begin{cases} [x_i], & \text{if } i \leq k; \\ [x_1], & \text{otherwise.} \end{cases}$$

for $1 \leq i \leq m$.

We will identify each element $x \in \mathcal{A}$ with the one-element string $[x] \in \Gamma(\mathcal{A})$. Then the structure $\mathcal{A}$ is an automorphism base of $\Gamma(\mathcal{A})$ (due to $[x_1, \dots, x_k] = [x_1] * \cdots * [x_k]$), and so Γ is a p.r. functor.

We now need to argue that Γ has a p.r. pseudo-inverse. First, note that

$$|y| = i \iff y = g_1(y) * \cdots * g_i(y)$$

and the right-hand side can be checked in a p.r. way in any copy. The pseudo-inverse of Γ should just be the functor which takes $\mathcal{U} \cong \Gamma(\mathcal{A})$ to the structure whose domain is the set of strings of length one with f_k defined by $f_k(x_1, \dots, x_k) = g_0(x_1 * \cdots * x_k)$. This is a computable functor, but to see that it is a p.r. functional, we need to see that we can find new strings of $\mathcal{U}$ which have length 1 in a punctual way.

Suppose $\mathcal{U} \cong \Gamma(\mathcal{A})$ for punctual structures

$$\mathcal{A} = \langle \omega, f_i^{\mathcal{A}} : 1 \leq i \leq m \rangle \text{ and } \mathcal{U} = \langle \omega, *^{\mathcal{U}}, g_i^{\mathcal{U}} : 0 \leq i \leq m \rangle.$$

Define functions p, q and r by primitive recursion: $p(0) = 0, q(0) = 1, r(0) = g_1^{\mathcal{U}}(0)$, and for $s > 0$

$$p(s) = (\mu x)[(\exists i_{1 \leq i \leq m})(\forall t_{t<s})[g_i^{\mathcal{U}}(x) \neq r(t)]],$$
$$q(s) = (\mu i_{1 \leq i \leq m})[(\forall t_{t<s})[g_i^{\mathcal{U}}(p(s)) \neq r(t)]],$$
$$r(s) = g_{q(s)}^{\mathcal{U}}(p(s)).$$

In other words, in $p(s)$ we search for the least element of $\mathcal{U}$ which (as a "string") contains a new symbol (one-element "string") not belonging to $\{r(0), r(1), \dots, r(s-1)\}$. Then $r(s)$ is a first such symbol. Since lengths of "strings" are bounded by m, the search in the μx-operator is restricted by $1+s+s^2+\cdots+s^m$, so the functions p, q and r are primitive recursive. Note also that p is a non-decreasing function such that $p(s) \geq \left[\frac{s}{m}\right]$ (p jumps at least every m-th stage), and r is a bijection from ω onto the image of $g_i^{\mathcal{U}}$ (i.e., onto the set of one-element "strings" in $\mathcal{U}$). Moreover, since $g_1^{\mathcal{U}}(r(s)) = r(s) = g_{q(s)}^{\mathcal{U}}(p(s))$ we have $p(s) \leq r(s)$ by the definition of $p(s)$, and, hence, if the value $r^{-1}(y)$ exists then

$$\left[\frac{r^{-1}(y)}{m}\right] \leq p(r^{-1}(y)) \leq y.$$

We conclude that r^{-1} is also primitive recursive.

Now we can define a punctual structure $\mathcal{B} = \langle \omega, f_i^{\mathcal{B}} : 1 \leq i \leq m \rangle \cong \mathcal{A}$ by

$$f_i^{\mathcal{B}}(x_1, x_2, \dots, x_i) = r^{-1}(g_0^{\mathcal{U}}(r(x_1) *^{\mathcal{U}} r(x_2) *^{\mathcal{U}} \cdots *^{\mathcal{U}} r(x_i))).$$

The pseudo-inverse of Γ is the functor Θ that maps $\mathcal{U}$ to $\mathcal{B}$. It is easy to see that $\Theta \circ \Gamma$ is isomorphic to the identity. It remains to check that $\Gamma \circ \Theta$ is isomorphic to the identity. To see this, we need to show that there are, uniformly, primitive recursive isomorphisms in each direction between $\mathcal{U}$ and $\Gamma(\Theta(\mathcal{U})) = \Gamma(\mathcal{B})$. The primitive recursive function

$$H([x_1, x_2, \dots, x_i]) = r(x_1) *^{\mathcal{U}} r(x_2) *^{\mathcal{U}} \cdots *^{\mathcal{U}} r(x_i) \text{ for } 1 \leq i \leq m,$$

will be an isomorphism from $\Gamma(\mathcal{B})$ onto $\mathcal{U}$. To see that H^{-1} is also primitive recursive it is enough to note that the "length" $|y|^{\mathcal{U}}$ can be computed as

$$|y|^{\mathcal{U}} = (\mu i_{1 \leq i \leq m})[y = g_1^{\mathcal{U}}(y) *^{\mathcal{U}} \cdots *^{\mathcal{U}} g_i^{\mathcal{U}}(y)],$$

and then

$$H^{-1}(y) = [r^{-1}(g_1^{\mathcal{U}}(y)), r^{-1}(g_2^{\mathcal{U}}(y)), \dots, r^{-1}(g_{|y|^{\mathcal{U}}}^{\mathcal{U}}(y))].$$

$\Gamma(\mathcal{A})$ is not yet a structure with only one binary function symbol, so we still need to reduce the language further.

Step 3. To prove the theorem we need remove the unary functions g_i, $0 \leq i \leq m$, from the signature of $\Gamma(A)$ emulating them using only one binary function $*$ defined above; this can be done by further enriching the structure by new elements. To do so we add into $\Gamma(A)$ new elements

$$c_0, c_1, \dots, c_m$$

such that

$$c_i * c_j = c_{\min(m, i+j+1)} \text{ for } 0 \leq i, j \leq m,$$

$$x * c_i = c_0, \text{ for } 0 \leq i \leq m, x \neq c_0, c_1, \ldots, c_m,$$
$$c_i * x = g_i(x), \text{ for } 0 \leq i \leq m, x \neq c_0, c_1, \ldots, c_m$$

The operation $*$ between the old and the new elements is no longer associative.

It is clear that each of the new elements $c_0, c_1, \ldots, c_m$ is definable; for instance, c_0 is the unique element such that $x * c_0 = c_0$ for at least $m + 2$ different x; $c_1 = c_0 * c_0$; $c_2 = c_1 * c_0$; etc. Notice that the searches that naturally correspond to all these definitions are bounded. Therefore this is another p.r. reduction and we can remove g_i, $0 \leq i \leq m$, from the signature of $\Gamma(\mathcal{A})$.

3. Punctually categorical graphs: Proof of Theorem 1.8

The implication (2) $\implies$ (1) in Theorem 1.8 is trivial. We prove (1) $\implies$ (2). We start with the simplest case in which G is essentially locally finite. For now, we focus only on proving that G is either almost a clique or almost an anti-clique; we will establish its homogeneity with respect to the finitely many exceptional nodes later.

Proposition 3.1. *Suppose a punctually categorical graph G becomes locally finite after removing finitely many vertices. Then G becomes an anti-clique after removing finitely many vertices.*

Proof. Let G be the graph. Name the finite tuple $\bar{a} = (a_1, a_2, \ldots)$ of exceptional nodes as constants, and let G' be $G \setminus \bar{a}$. Each $x \in G'$ has an *$\bar{a}$-colour*, which is the configuration of the form

$$(x, a_1) \in E(G), (x, a_2) \notin E(G), \ldots,$$

that is, the relation induced by the edge relation w.r.t. the fixed finite tuple.

Consider the cases below. In each case we describe the strategy for building a punctual $H \cong G$ such that $H \not\cong_p G$, where $\cong$ stands for a primitive recursive isomorphism with primitive recursive inverse. (We call such isomorphisms punctual.)

Case 1. The graph G' has infinitely many connected components.

- Case 1.1 All but finitely many components in G' are singletons, and all components are finite. In this case G' (thus, G as well) becomes a totally disconnected set after removing finitely many nodes.
- Case 1.2 Either there is an infinite connected component or there are infinitely many non-trivial (non-singleton) connected components in G'. There must exist an infinite independent $\bar{a}$-monochromatic set. Build a graph H while diagonalizing one by one against primitive recursive isomorphisms with G. To diagonalize against a particular isomorphism p_e, in H, enumerate elements from the monochromatic independent set very quickly (though we cannot yet choose images in G for these elements) and wait for $p_e : G \to H$ to converge on more nodes x which are connected to at least one other node in G, but currently keep such nodes out of H. At some point, p_e must converge and show a mistake. Then we must include in H some of the nodes from G which we have so far omitted while also choosing images in G for some of the elements in H from the monochromatic independent set. After this, we move on to the next primitive recursive isomorphism. So this case is impossible.

Case 2. The graph G' has finitely many connected components. In this case at least one of the components must be infinite, and must contain an infinite $\bar{a}$-monochromatic set. We diagonalise against a pair (p_e, q_e), where q_e is the intended inverse of p_e. Initially let H copy G and wait for $p_e : H \to G$ to converge on some node x within one such infinite component,

$$p_e(x) = h \in H.$$

As soon as this happens, introduce a new extra $h' \in H$ whose $\bar{a}$-colour is the same as the $\bar{a}$-colour of the infinite $\bar{a}$-monochromatic set in the component. Wait for $q_e(h') \downarrow = x' \in G$, and wait for x' to be put into one of the finitely many connected components in G'. While we wait we let H copy G, but currently keep h' out of the range of the (computable) isomorphism that we are constructing.

- If the component of x' it is *not* the same as the component of x, then map h' to the first found fresh and unused element in the component of x in G' which has the same $\bar{a}$-colour. We have diagonalised.
- If the component of x' is the same as the component of x, then let $x, x_1, \ldots, x_k, x'$ be a path that witnesses this. Wait for p_e to converge on each of the x_i in the paths, but keep h' disconnected from h. After this is done, we have diagonalised. It remains to introduce an image of h' in G, this is done as above.

Note that the strategies differ in different cases. □

To understand the case which is far from being locally finite we will need the technical proposition below. It says that if $deg(x) = \infty$ then the set $N(x) = \{z : (z, x) \in E(G)\}$, which we call the 1-neighbourhood (1-nbhd) of x, must be growing punctually.

Proposition 3.2. *Suppose G is a punctually categorical graph, and assume $deg(x) = \infty$ for some $x \in G$. Then the 1-nbhd of x must be rapidly growing, i.e., there exists a primitive recursive time-function f such that*

$$|N(x)\,[s]| < |N(x)\,[f(s)]|,$$

for every stage s of the construction.

Proof idea. We write $|X|$ for the cardinality of X. Imagine that x is the only vertex of G with the property $deg(x) = \infty$, and assume $N(x)$ is very slowly growing, i.e., there is no primitive recursive bound on the stage at which the nth vertex appears in $N(x)$. In this simple case the strategy is straightforward. Build a copy B of G which is essentially identical to G but with $|N_B(x)| = |N_G(x)| - 1$ at every stage. Any isomorphism must match the points of infinite degree, and we know the graph has only one such point. We also know that "most of the time" G puts points into $G \setminus N(x)$, and therefore in B we can eventually delay one element from appearing in the 1-nbhd of x and still keep B punctual. All we need to do is to wait until $p : G \to B$ is diagonalised on the extra vertex that G has in $N(x)$ when compared with B. Note that we must eventually succeed, for otherwise we could use p to extract a primitive recursive bound on the speed of growth of $N(x)$ in G.

When G has many vertices of infinite degree and is not rigid, we have to consider 2 copies, A and B, of G and look at two vertices in G. This is necessary because $p : B \to A$ does not have to map the natural version of x in B to the natural version of x in A. So suppose A is copying G via ψ and B via ϕ (both are defined by us), and suppose we are trying to diagonalise against a pair (p, q), where $p : B \to A$ and (supposedly) $p^{-1} = q$.

The idea is to either keep $|N_A(p\phi(x))| < |N_B(\phi(x))|$ or $|N_A(p\phi(x))| > |N_B(\phi(x))|$ for as long as possible, and use either p or q to press the opponent to grow the respective neighbourhoods in G. It is also crucial to avoid $|N_A(p\phi(x))| = |N_B(m\phi(x))|$ at all costs, because in this situation we cannot "press" the opponent. We informally explain how we "press" below.

For example, suppose at stage s have $|N_A(p\phi(x))| < |N_B(\phi(x))|$. Then evaluate p on $N_B(\phi(x))[s]$; the opponent must grow $N_G(\psi^{-1}p\phi(x))$ in G. The trick here is that we do not make the 1-neighbourhood equal, but rather delay at most one point and press the opponent to grow $N_G(\psi^{-1}p\phi(x))$ 1-point larger than $N_B(\phi(x))[s]$. So one point appears in G and makes the two 1-nbhds look equal; however, we keep this point out of our structure A and wait for another point to appear in $N_G(\psi^{-1}p\phi(x))$. This is fine to delay one point from the diagram of A. Just consider the next point u in G. If $u \notin N_G(\psi^{-1}p\phi(x))$ then copy it into A. If it is in $N_G(\psi^{-1}p\phi(x))$ then this is exactly what we needed. But recall that we challenged the opponent by computing p on the currently larger $N_B(\phi(x))$. The opponent must respond by enumerating more points into $N_G(\psi^{-1}p\phi(x))$, and it must do so within the time bound of p, for otherwise p is not an isomorphism. Thus, unless $N_G(x)$ (thus, $N_B(\phi(x))$) grows within the time needed for $p(N_B(\phi(x))[s])$ to converge, we will be able to make $|N_A(p\phi(x))| > |N_B(\phi(x))|$. But then we can evaluate q on $N_A(p\phi(x))$ and similalry press the opponent to grow $N_B(\phi(x))$ by extending $N_G(x)$ in G; this was our ultimate goal. In the worst case scenario the time bound can be extracted from and $q(N_A(p\phi(x))[t])$, were t depends on the convergence time for $p(N_B(\phi(x))[s])$, making the described above process punctual.

The sketch above describes the worst case scenario, but there will be various cases which also have to be incorporated into the formal argument below. To maintain the inequality $|N_A(p\phi(x))| \neq |N_B(\phi(x))|$

at every stage we will have to consider the cases when either $N_A(p\phi(x))$ or $N_B(\phi(x))$ starts growing faster than anticipated. Also, $|N_A(p\phi(x))|$ and $|N_B(\phi(x))|$ do not have to differ by only one element at a given stage. In all these cases we have even more advantage over the opponent, however, considering such cases will significantly increase the combinatorial complexity of the formal argument below, making it not that easy to follow. At this stage the reader may want to see the formal details.

Proof of Prop. 3.2. For any vertex x of a graph G, define its 1-neighbourhood (1-nbhd) to be the set $N(x) = \{y : (x, y) \in E(G)\} \cup \{x\}$, i.e., the set of points at distance ≤ 1 from x. We write $|X|$ for the cardinality of X. Using two auxiliary punctual copies, A and B, of G we will produce a primitive recursive bound on the stages at which new points get enumerated into $N(x)$. The copy A will be copying G via computable isomorphism ψ, and B via ϕ. The isomorphisms ψ and ϕ will be "natural" (i.e., the identity) at most of the stages, but with some local modifications described by the strategy below.

We will attempt to diagonalise against each pair (p, q), where $p : B \to A$ is trying to illustrate that it is a punctual isomorphism with $p^{-1} = q$. We must fail to diagonalise against one (least) such pair which actually does represent a punctual isomorphism; this will allow us to extract a primitive recursive bound for the speed of growth of $N(x)$.

Write x' for $\phi(x) \in B$. Assume B is currently naturally copying G via ϕ, and A is currently naturally copying G via ψ. Without loss of generality, assume $p(x') \downarrow$ and $q(p(x')) \downarrow = x'$. At every stage s we monitor p and q on the first s inputs and check that $q = p^{-1}$ and both preserve the edge relation. If this property does not hold, then we immediately halt the module described below and initiate the module that looks at the next highest priority pair (p', q').

The diagonalisation module for (p, q).

i. Make sure that $|N(x')| \neq |N(p(x'))|$. If the inequality already holds then do nothing and proceed to ii. Otherwise, if currently $|N(x')| = |N(p(x'))|$, do the following. Wait for either a fresh y to be put into $N(x)$ in G or a fresh z to appear in $N(\psi^{-1}(p(x')))$ (or both). (While waiting let A and B copy G.) The following cases are possible:
   (a) Such a y appears in $N(x)$ but this same element is not in $N(\psi^{-1}(p(x')))$, then we have $|N(x')| > |N(p(x'))|$.
   (b) A fresh z appears in $N(\psi^{-1}(p(x')))$ and $z \notin N(x)$, then we have achieved $|N(x')| < |N(p(x'))|$.
   (c) We have found $y \in N(\psi^{-1}(p(x'))) \cap N(x)$. Then put the natural ψ-image of y into A, but currently keep y out of the domain of ϕ. We currently have $|N(x')| < |N(p(x'))|$.

   Depending on whether $|N(x')| < |N(p(x'))|$ or $|N(x')| > |N(p(x'))|$, go to ii. or iii. below, respectively.

ii. Suppose $|N(x')[s]| < |N(p(x'))[s]|$. (In this case perhaps there is one element y of G currently intentionally kept outside of the domain of ϕ.)
   (a) Start waiting for q to converge on all elements in $N(p(x'))[s]$.
   (b) Consider the next $k = (|N(p(x'))| - |N(x')|) + 1$ points in G currently outside of the domain of ψ (including y in the notation above, if it exists).
   (c) Pick a non-empty subset S of the set of these k points such that one of the two conditions below holds (if such a subset exists):
      (1) The natural extension of ψ to S makes $N(x')$ grow in B, and the inequality $|N(x')| < |N(p(x')|$ is preserved under the natural extension of ψ and ϕ to S. (If y is defined then it must be the case that $N(p(x'))$ also grows.) If S contains a $y' \neq y$ such that $y' \in N(\psi^{-1}(p(x'))) \cap N(x)$, then define $\phi(y)$ and $\psi(y')$ naturally, but delay the definition of $\phi(y')$. Then restart ii. with the new values of $|N(x')|$ and $|N(p(x'))|$, and let y' play the role of a "new" y. (We shall abuse our notation and identify this y' with y at later stages. Note that in this case $N(x)$ has increased in size in G before q to converged on all elements in $N(p(x'))[s]$.)
      (2) The natural extension of ψ and ϕ to S flips the inequality $|N(x')| < |N(p(x')|$ into $|N(x')| > |N(p(x')|$. (If y is defined then we require that $y \in S$ and $N(x)$ grows

by at least two extra points, i.e., $S \cap N(x)$ contains more than just y.) In this case extend ψ and ϕ to S naturally and go to iii. below.

(d) There is no subset S having one of the two properties (1) and (2) above. Then one of the two possibilities must hold:

(1) If y is undefined, then there is z among the k elements that we consider with the property $z \in N(x)$, and furthermore $z \notin N(\psi^{-1}(p(x')))$. In particular, adding the natural ϕ- and ψ-images to the respective structures for any subset $S \ni z$ of these k elements will result in $|N(x') = N(p(x'))|$. Then extend ϕ and ψ to all the k elements except for z and return to the second clause of ii. with z playing the role of y and with new fresh k elements that contain $y = z$. (In particular, keep z out of the domain of ϕ.)

(2) y is defined. Since none of the above cases apply, there must be no points among the considered $k = 2$ points that belong to $N(x)$ with the exception of y. Naturally extend ϕ and ψ to these k points, but keep y out of the domain of ϕ. Also, update the value of k if necessary if $N(p(x'))$ has increased, *but keep waiting for q to converge merely on all elements of (the potentially smaller) $N(p(x'))[s]$*, where s was the stage when ii. was initiated. Then return to (b) of ii. with new fresh k elements.

iii. Suppose $|N(x')[s]| > |N(p(x'))[s]$. This case is very similar to ii. above, but without the extra complication related to the auxiliary element y.

(a) Start waiting for p to converge on $N(x')[s]$.

(b) Whenever a new point enters $N(x)$, naturally define ϕ and ψ on x and extend ϕ and ψ to the first found extra point currently outside of their domain. Go to iii. with these new parameters. (Note that $|N(x')| > |N(p(x'))|$ still holds.) *In the substeps below we assume that new elements do not enter $N(x)$. If they do, (b) acts immediately (but see (f) for a mild modification).*

(c) Consider the first unused $(|N(x')|[s] - |N(p(x'))|[s]) + 1 = k$ many points in the domain of G.

(d) If all of these points belong to $\psi^{-1}(N(p(x')))$, then adjoin these points to $N(p(x'))$ and define ψ on these points naturally. In this case we end up with $|N(x')| < |N(p(x'))|$ (recall that $N(x')$ is not growing, see above). Go to ii.

(e) If there are less than $k - 1$ points that belong to $\psi^{-1}(N(p(x')))$ among these k points, then extend ϕ and ψ to these points, thus (perhaps) growing $N(p(x'))$. Since $N(x)$ was assumed to not grow, we decrease $|N(x')| - |N(p(x'))|$ and thus k as well. The choice of k (and the k-1 points) implies that $|N(x')| > |N(p(x'))|$ is preserved. Repeat for the next k points in G, with the new value for k.

(f) If there are exactly $k - 1$ points (among the k points) that belong to $\psi^{-1}(N(p(x')))$, then pick one such point z and temporarily keep it out of the domains of ψ and ϕ. Naturally define ϕ and ψ on the rest of the k points. (Note that we will end up with $|N(x')| - |N(p(x'))| = 1$.) After this is done, start waiting for one more point to enter $\psi^{-1}N(p(x'))$ (while waiting, keep building A and B elsewhere). If a new point enters $N(x)$ while we are waiting, then define ϕ and ψ on z and act as prescribed in (b) and note that $|N(x')| > |N(p(x'))|$ still holds in this case. Otherwise, as soon as such a point is found, naturally define both ϕ and ψ on both this point and on z which was kept outside of their domains. As the result of this action, we will have $|N(x')| < |N(p(x'))|$. Go to ii.

This finishes the description of the diagonalisation module.

Construction. Let the modules act one after another, according to some effective listing of pairs (p, q). When the highest priority module succeeds in diagonalising against its pair of functions, we remove the restraints of the module by extending ϕ (or ψ) to the point which was currently kept out of its domain, and then we initiate the next highest priority module, etc.

Verification. Observe that all delays on the enumeration of the diagram of A and B imposed by the module are bounded and did not depend on the time needed to compute p or q. Furthermore, we

explicitly ensured that at most one new vertex is put into A and B with a delay that depends only on the diagram of G and the stage of the construction (e.g., the parameter k) in a primitive recursive fashion. Thus, both A and B are (infinite) punctual graphs. Also, any element that could be delayed from being mapped to B via ϕ will eventually be put into the domain of ϕ, and similarly for A and ψ. This is because either (p, q) will prove that it is not an isomorphism in finite time and thus the restraint of the basic module will be removed, or the clause ii. or iii will eventually finish its action, which also always results in the restraint being dropped. By their definition, ϕ and ψ are (computable) isomorphisms. Thus, regardless of the outcomes of the modules, both A and B are punctual copies of G computably isomorphic to G via ψ and ϕ, respectively.

Recall that at every stage we check whether $p = q^{-1}$ on the first few inputs of p and q, and that p (and q) looks like an isomorphism on these inputs. Therefore, if (p, q) does not represent a punctual isomorphism then the module corresponding to (p, q) will eventually halt. There must exist a pair (p, q) which represents an actual punctual isomorphism between B and A; assume (p, q) is the highest priority such pair.

The adjustment on stage i. results in either ii. or iii. becoming active. Note that i. will eventually finish its action because the degree of x is infinite, and therefore at least one of the conditions in i. will eventually be satisfied (more specifically, a new vertex will eventually enter $N(x)$).

Since (p, q) represents an isomorphism, then the condition $|N(x')| > |N(p(x'))|$ implies that a new element must appear in $N(p(x'))$ before p halts on all elements currently in $N(x')$, and similarly for the case $|N(x')| < |N(p(x'))|$. We list the possible scenaria in ii. of the module below.

(1) If the action happens at (1) of (c), then this means that $N(x)$ has grown by one extra element within the time needed to compute q on on all elements in $N(p(x'))[s]$, where s is the stage at which ii. was first initiated (and is *not* the current stage). In this case we stay in ii.
(2) Similarly, if the action happens at (2) of (c) then $N(x)$ must have increased within the same as above time bound. In this case we go to iii.
(3) Note that by the choice of k there will always be some action at (d) of ii., in the sense that some new points will be put into both A and B. (We note that "putting new points" really means "defining the diagram on a longer segment of ω".)

We argue that either (1) or (2) of (c) will eventually act. Since q must be a primitive recursive isomorphism, it must eventually converge on $N(p(x'))[s]$, and the images must be in $N(x')$. Thus, $N(x')$ must be forced to grow by G; in other words, G must start enumerating points into $N(x)$. The potential delay/restraint of at most one point from the domain of $\psi : G \to B$ makes no difference. We could have promised to keep at most one point out of B, this merely a bounded delay. Thus, G must respond by giving another point in $N(x)$, otherwise we will diagonalise against (p, q). As the result, we wither act in (1) of (c) or we act in (2) of (c). In either case the module ensures that $N(x)$ has grown by at least one point within the time required for q to converge on the finite set $N(p(x'))[s]$.

We now look at iii. Similarly to ii., we set a time bound at (a) of iii. If we act at (b) then $N(x)$ must have increased within the time bound. If we act at (d), then within the time bound we will see the size $N(p(x'))$ grow beyond the size of $N(x')$, *but it is not obvious how this gives the bound on the next stage at which $N(x)$ grows.* Indeed, within the bound set by (a) of iii., $N(x)$ may have stayed the same. But it is sufficient to note that in (d) we will switch to ii., which itself will pick a bound based on q and a finite subset of the currently defined part of A (namely, $N(p(x'))$). The size m of the part of A which will have been enumerated by then depends on the stage in a primitive recursive way. *Therefore, when we initiate iii., the time bound on the growth of $N(x)$ to be within the time needed for q to converge on the first m elements of A*, where m primitively recursively depends on the stage s and the time needed for the computation of p on $N(x)[s]$.

If we act in (e) then we call for iii., but with adjusted parameters, but with the same time bound that was set above. In (f), either $N(x)$ increases within the time bound, or we go to ii. Similarly to (d), it is crucial that the actual bound that we impose on the speed of growth of $N(x)$ is different from the bound that is used by the clause iii.

It follows from the analysis above that $N(x)$ will be forced to grow, and furthermore there is a primitive recursive bound on the stages at which $N(x)$ increases. □

The rest of the proof will be a sequence of (neat) observations making use of the main technical proposition above. We note that all the results that we obtained so far apply for the graph-theoretic complement of G (the same vertices under the non-edge relation).

Proposition 3.3. *Suppose G is punctually categorical and $deg(x) = \infty$ for some $x \in G$. Then x is connected to a.e. vertex in G.*

Proof. Suppose that $deg(x) = \infty$. By Proposition 3.2 we see that $N(x)$ is prompt. Then we can produce a punctual copy H of G and an element $y \in H$ which is the image of x under some isomorphism, such that $H - N_H(y)$ is not punctual in the sense of Proposition 3.2. Then Proposition 3.2 applied to the graph-theoretic complement of H implies that $H - N_H(y)$ is finite. Thus x is connected to almost every vertex.

To build H, as usual we copy G while delaying some elements. Fix a particular image $y \in H$ of x. We diagonalize against primitive recursive bounds on the speed of growth of $H - N_H(y)$. At some point of the construction, H has copied some finite portion of G and we must diagonalize against the next primitive recursive bound p. To do this, we copy only elements of $N(x)$ into H. We must eventually see a disagreement with p. Once we have diagonalized against the bound, copy all of the skipped elements into H and move on to diagonalize against the next bound. □

Proposition 3.4. *Suppose G has infinitely many vertices of infinite degree. For each $n \in \omega$, there are only finitely many vertices of degree n. The same holds for co-degree n.*

Proof. Fix any $n + 1$ distinct vertices of infinite degree. By the previous proposition, almost every vertex in G is adjacent to *all* of them, and thus has degree at least $n + 1$. □

Proposition 3.4 implies that—provided that G has infinitely many vertices of infinite degree and infinitely many vertices of finite degree—the automorphism orbit of any vertex in G must be finite.

Proposition 3.5. *Suppose G has infinitely many vertices of infinite degree and infinitely many vertices of finite degree. For each $v \in G$, consider the graph $(G - v)$ obtained by removing the vertex v from G. Then $G \not\cong (G - v)$.*

Proof. Consider the case when $deg(v) = n < \infty$, the case of co-finite degree is symmetric. Notice that for any x in $(G - v)$, its degree in $(G - v)$ is either the same as its degree in G, or is one less. And the latter can only happen if x is adjacent to v.

Suppose v has degree n. By Proposition 3.4, there are only finitely many vertices of degree n. If G is isomorphic to (G-v), some y_0 in G drops from degree $n + 1$ to degree n. But then there must also be some y_1 which drops from degree $n + 2$ to degree $n + 1$, etc. But each of these y_i is adjacent to v, contrary to the assumption that v has finite degree. □

We are ready to finish the proof of Theorem 1.8. Suppose that G is punctually categorical. First, we argue that G has only finitely many vertices of finite degree or only finitely many vertices of infinite degree. Indeed, suppose to the contrary that G is not like that, and so $G \not\cong (G - v)$ for any $v \in G$ by Proposition 3.5. In this case it is easy to show that G cannot be punctually categorical, as follows. To diagonalise against a pair (p, q), we simply delay one vertex v in an auxiliary copy that we build. We wait for (p, q) to illustrate a disagreement, and then we can put v into our copy.

Then by Proposition 3.1 applied to G or to the graph-theoretic complement of G, G becomes a clique or an anti-clique after removing finitely many vertices $\bar{v}$. By Proposition 3.3, each of these vertices is either connected to almost every vertex in G, or is not connected to almost every vertex in G. Let $\bar{v}$ be the finite set of exceptional nodes from $(G - \bar{u})$. Since $(G - \bar{u})$ is either a clique or an anti-clique, each node in $\bar{v}$ is connected to every or to no vertices of $(G - \bar{u})$. Thus $(G - \bar{u}\bar{v})$ satisfies the theorem.

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.

[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.

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

[DM08] R. Downey and A. Montalbán. The isomorphism problem for torsion-free abelian groups is analytic complete. *J. Algebra*, 320(6):2291–2300, 2008.

[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.

[EGN+98a] Yu. L. Ershov, S. S. Goncharov, A. Nerode, J. B. Remmel, and V. W. Marek, editors. *Handbook of recursive mathematics. Vol. 1*, volume 138 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, 1998. Recursive model theory.

[EGN+98b] Yu. L. Ershov, S. S. Goncharov, A. Nerode, J. B. Remmel, and V. W. Marek, editors. *Handbook of recursive mathematics. Vol. 2*, volume 139 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, 1998. Recursive algebra, analysis and combinatorics.

[GD80] S. S. Gončarov and V. D. Dzgoev. Autostability of models. *Algebra i Logika*, 19(1):45–58, 132, 1980.

[GK02] S. Goncharov and J. Knight. Computable structure and antistructure theorems. *Algebra Logika*, 41(6):639–681, 757, 2002.

[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.

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

[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.

[HTMM15] Matthew Harrison-Trainor, Alexander Melnikov, and Antonio Montalbán. Independence in computable algebra. *J. Algebra*, 443:441–468, 2015.

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

[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.

[LaR77] P. LaRoche. Recursively presented boolean algebras. *Notices AMS*, 24:552–553, 1977.

[Lav63] I. A. Lavrov. The effective non-separability of the set of identically true formulae and the set of finitely refutable formulae for certain elementary theories. *Algebra i Logika Sem.*, 2(1):5–18, 1963.

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

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

[Mel17a] Alexander Melnikov. Computable topological groups and pontryagin duality. page 1, 08 2017.

[Mel17b] 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.

[MM18] Alexander Melnikov and Antonio Montalbán. Computable Polish group actions. *J. Symb. Log.*, 83(2):443–460, 2018.

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

[Mon13] Antonio Montalbán. A computability theoretic equivalent to Vaught's conjecture. *Adv. Math.*, 235:56–73, 2013.

[MPSS] R. Miller, B. Poonen, H. Schoutens, and A. Shlapentokh. A computable functor from graphs to fields. To appear.

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

[Rem81] J. B. Remmel. Recursively categorical linear orderings. *Proc. Amer. Math. Soc.*, 83(2):387–391, 1981.
[Ric81] Linda Jean Richter. Degrees of structures. *J. Symbolic Logic*, 46(4):723–731, 1981.
[RS] M. Rabin and D. Scott. The undecidability of some simple theories. Unpublished notes.
[Tsa11] Todor Tsankov. The additive group of the rationals does not have an automatic presentation. *J. Symbolic Logic*, 76(4):1341–1351, 2011.
[vdW30] B. van der Waerden. Eine Bemerkung über die Unzerlegbarkeit von Polynomen. *Math. Ann.*, 102(1):738–739, 1930.

School of Mathematics and Statistics, Victoria University of Wellington, PO Box 600, Wellington, New Zealand.
E-mail address: rod.downey@vuw.ac.nz

School of Mathematics and Statistics, Victoria University of Wellington, PO Box 600, Wellington, New Zealand.
E-mail address: matthew.harrisontrainor@vuw.ac.nz

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

School of Mathematics and Statistics, Victoria University of Wellington, PO Box 600, Wellington, New Zealand.
E-mail address: dan.turetsky@vuw.ac.nz