Relativizing Computable Categoricity*

Rod Downey, Matthew Harrison-Trainor, and Alexander Melnikov

November 17, 2020

Abstract

A recent thread in computable structure theory has been the investigation of computable structures after relativizing, the key idea being that facts which are true for algebraic/structural reasons tend to relativize. On the other hand, there are pathological examples such as a structure which is computably categorical but not relatively computably categorical; but such behaviour must eventually stabilize, as for example a structure is either computably categorical relative to all degrees above $\mathbf{0}''$ or not computably categorical relative to all degrees above $\mathbf{0}''$. But what can happen in between? We show a surprising result: there is a structure which alternates between being computably categorical and not computably categorical relative to an infinite increasing sequence of c.e. degrees.

1 Introduction

In classical mathematics, a principal program is the classification of structures like groups, rings, and C*-algebras up to isomorphism. *Computable* mathematics is concerned with *algorithmically* presented structures, and their algorithmic properties. In general, a structure is *computable* if it is given with universe and atomic diagram both computable. Demanding that the structures are represented algorithmically so that basic operations are computable, and that solutions be computable, tends to mean that questions require much finer-grained analytic and algebraic techniques. For instance, finding a basis for a variety is much more involved than a proof that one exists.

A consequence of such considerations is that classical isomorphism becomes refined into *computable* (or otherwise computational) isomorphism as a classification tool. Starting with the seminal work of Maltsev [Mal61], a lot of effort has been put into understanding the relationship between classical and computational isomorphism both in general and for particular classes of structures (such as groups).

A basic definition in this area is the following.

Definition 1.1. A computable structure $\mathcal{A}$ is computably categorical if for every computable copy $\mathcal{B}$ of $\mathcal{A}$, there is a computable isomorphism between $\mathcal{A}$ and $\mathcal{B}$.

*We thank Denis Hirschfeldt for many fruitful discussions.

Starting with Maltsev [Mal61], there have been a large number of results characterising computable categoricity in natural classes of structures; see [LaR77, Smi81, Gon80a, Rem81] and also the recent papers [MN18, HTMM15]. Research on computable categoricity tended to split into related approaches. One was motivated by *algebra* and looked for algebraically natural ways of classifying computably categorical structures. For instance, the results of Goncharov and Remmel show that a linear ordering is computably categorical if and only if it has a finite number of adjacencies, a Boolean algebra is computably categorical if and only if it has only finitely many atoms, and a torsion-free abelian group is computably categorical if and only if its rank is finite. After several decades of research it has become evident that a purely algebraic characterisation of computable categoricity in a class should be possible only when computable categoricity is equivalent to *relative computable categoricity* in the class:

Definition 1.2. A computable structure $\mathcal{A}$ is relatively computably categorical if for every not necessarily computable copy $\mathcal{B}$ of $\mathcal{A}$, there is a $\mathcal{B}$-computable isomorphism between $\mathcal{A}$ and $\mathcal{B}$.

The intuition is that if a structure is computably categorical for a purely algebraic reason then this should be true under *relativization*. The other related approach to computable categoricity is based on effectivising model theory. Scott's Isomorphism Theorem [Sco65] says that a countable structure $\mathcal{A}$ is specified up to isomorphism by an infinitary sentence in the language $\mathcal{L}_{\omega_1\omega}$, a Scott sentence.

Working independently, Ash, Knight, Manasse, and Slaman [AKMS89], Chisholm [Chi90] and Ventsov [Ven92] formally clarified the relationship between the algebraic and the model-theoretical using *Scott families*. They showed that a structure is relatively computably categorical if and only if there is a computably enumerable family of first-order existential quantifiers describing automorphism orbits of tuples in the structure; in modern terms, if and only if the structure has a Σ_1 Scott family [AK00].

If a structure possesses a computable Scott family, then it is computably categorical via a back-and-forth proof. However the converse is not true. Goncharov (see [EG00]) constructed a computably categorical structure with no Σ_1 Scott family. Furthermore, there is no reasonable characterization of the computably categorical structures in general [DKL$^+$15]; formally, their index set is Π^1_1-complete.

Does this mean that computable categoricity is a wrong notion? Is there any hope of effectivizing Scott's Isomorphism Theorem and capturing computable categoricity using syntax? When we cannot characterise algebraic structures with a certain property it is natural to seek extra conditions which would make such a characterisation possible. Goncharov [Gon77, Gon80b] showed that this is indeed possible if the underlying structure obeys a stronger computational hypothesis than only being computable, namely it is 2-decidable:

Theorem 1.3 (Goncharov [Gon80b])**.** *If a structure is computably categorical and its $\forall\exists$ theory is decidable, then it is relatively computably categorical*

The proof of this result relativizes. Thus, if $\mathcal{A}$ is 2-decidable and computable relative to X, then $\mathcal{A}$ is relatively computably categorical relative to X, meaning that any two Y-computably presented structures are Y-computably isomorphic for $Y \geq_T X$.

Now suppose that we have a computable structure $\mathcal{A}$ which is computably categorical but not relatively computably categorical, so that there is some $X >_T \varnothing$ such that $\mathcal{A}$ is not computably categorical relative to X. A simple consequence of the above theorem of Goncharov is the following:

Fact 1.4. *If $\mathcal{A}$ is computable and computably categorical relative to some degree $\mathbf{d} \geq \mathbf{0}''$, then $\mathcal{A}$ has a $\mathbf{0}''$-computable Σ_1 Scott family.*[1] *(Hence $\mathcal{A}$ is computably categorical relative to every degree above $\mathbf{0}''$.)*

Proof. We know that $\mathcal{A}$ is 2-decidable relative to $\mathbf{d}$, so by the aforestated result of Goncharov [Gon80b] it has a $\mathbf{d}$-computable Σ_1 Scott family. Now knowing that $\mathcal{A}$ has a Σ_1 Scott family, we will find a $\mathbf{0}''$-computable such family. Given $\bar{a} \in \mathcal{A}$ and an existential formula $\varphi(\bar{x})$ true of $\bar{a}$, ask whether for every tuple $\bar{b} \in \mathcal{A}$ satisfying $\varphi(\bar{x})$ and every existential formula $\psi(\bar{x})$, $\psi(\bar{a})$ if and only if $\psi(\bar{b})$. If this is the case, then $\varphi(\bar{x})$ isolates the orbit of $\bar{a}$ and we can enumerate it into our $\mathbf{0}''$-computable Scott family. □

Thus, $\mathcal{A}$ perhaps goes from being computably categorical to not computably categorical relative to X for some $\varnothing <_T X <_T \varnothing''$ and then at degree $\mathbf{0}''$ the pathology disappears: either for all $\mathbf{d} \geq \mathbf{0}''$ we have that $\mathcal{A}$ is computably categorical relative to $\mathbf{d}$, or for all $\mathbf{d} \geq \mathbf{0}''$ we have that $\mathcal{A}$ is not computably categorical relative to $\mathbf{d}$.

The natural question we ask is the following.

Question 1.5. If $\mathcal{A}$ is computable, what might the sets

$$\{X : \mathcal{A} \text{ is computably categorical relative to } X\}$$

and

$$\{Y : \mathcal{A} \text{ is not computably categorical relative to } Y\}$$

look like?

It is of course most natural to suspect that any process of switching from computably categorical to not computably categorical is in some way monotonic, i.e., once the pathology disappears at a degree $\mathbf{a} > \mathbf{0}$ it never occurs again above $\mathbf{a}$. The result below was unexpected.

Theorem 1.6. *There is a computable structure $\mathcal{A}$ and c.e. degrees $0 = Y_0 <_T X_0 <_T Y_1 <_T X_1 <_T \cdots$ such that*

1. *$\mathcal{A}$ is computably categorical relative to Y_i for each i,*
2. *$\mathcal{A}$ is not computably categorical relative to X_i for each i,*
3. *$\mathcal{A}$ is relatively computably categorical relative to $\mathbf{0}'$.*

[1] Note that there exist examples which are computably categorical but not relatively Δ^0_α categorical, where α can be an arbitrary large computable ordinal [DKL$^+$15], or even hyperarithmetically categorical [Tur]. Fact 1.4 does not contradict these examples since a computably categorical structure does not have to be computably categorical relative to $\mathbf{d} \geq \mathbf{0}''$. The reader should not confuse relative categoricity with categoricity relative to a fixed oracle. We also note that the standard example of a structure which is computably categorical but not relatively computably categorical has a $\mathbf{0}'$-c.e. Σ_1 Scott family but no c.e. Σ_1 Scott family.

Although the theorem above illustrates that there is much more chaos below $\mathbf{0}'$ than anticipated, it could be that the situation is a lot better above $\mathbf{0}'$, but this is something to investigate in the future. It is *sometimes* better above $\mathbf{0}'$ since Downey, Kach, Lempp and Turetsky showed that if $\mathcal{A}$ is 1-decidable, then $\mathcal{A}$ is relatively Δ^0_2 categorical. It is unclear if our example can be made 1-decidable.

The proof of Theorem 1.6 uses relatively standard ideas of "loops" and "isomorphism pressing" but it has some unusual features. It is of course a priority argument. The basic strategy is non-trivial but injury in the proof is merely finite. The guessing is complex enough to make some aspects of the construction resemble non-uniform infinite injury arguments, but the tree of strategies is not really helping to sort our the combinatorics. We chose to eliminate the tree of strategies from the construction. Thus, the reader should prepare for a non-standard proof.

The next section below is devoted to the proof of the main result. We finish the paper with a section which contains several observations some involving computable dimension, and a few open questions related to the topic of the paper.

2 Proof of Theorem 1.6

The structure $\mathcal{A}$ will be a directed graph consisting of infinitely many finite connected components. Each component will consist of either two, three, four, or five cycles sharing a single vertex, called the *root vertex* of the component. The root vertices can be identified as the only vertices of degree greater than two.

We build $\mathcal{A}$ stage-by-stage ensuring that it is computable. At the same time, we will build two sequences of uniformly c.e. sets $(X_i)_{i\in\omega}$ and $(Y_i)_{i\in\omega}$ and Turing reductions Ψ_k such that $\mathcal{B}_k = \Psi_k^{X_k}$ is a X_k-computable copy of $\mathcal{A}$ which is not X_k-computably isomorphic to $\mathcal{A}$. By enumerating elements into X_k, we can give Ψ_k permission to change $\mathcal{B}_k$.

For every set Z, let $(\mathcal{M}_i^Z)_{i\in\omega}$ a uniformly Z-computable list of the (possibly partial) Z-computable structures. To ensure that $\mathcal{A}$ is computably categorical relative to each Y_k, we meet the requirements:

$$\mathcal{S}_i^k : \text{If } \mathcal{M}_i^{Y_k} \cong \mathcal{A}, \text{ then } \mathcal{M}_i^{Y_k} \text{ is } Y_k\text{-computably isomorphic to } \mathcal{A}.$$

Recall that to make $\mathcal{A}$ not computably categorical relative to X_k, we build an X_k-computable structure $\mathcal{B}_k = \Psi_k^{X_k}$ which is not X_k-computably isomorphic to $\mathcal{A}$. To achieve this we meet the requirements:

$$\mathcal{R}_i^k : \Phi_i^{X_k} \colon \mathcal{A} \to \mathcal{B}_k = \Psi_k^{X_k} \text{ is not an isomorphism.}$$

Note that for a fixed k, the $\mathcal{R}$ requirements share the same $\mathcal{B}_k = \Psi_k^{X_k}$. We will build the X_i and Y_i by setting $Y_0 = \varnothing$, and

$$Y_{i+1} = X_i \oplus Y_{i+1}^* \qquad \text{and} \qquad X_i = Y_i \oplus X_i^*$$

where X_i^* and Y_i^* are c.e. sets to be defined by the construction. Thus we automatically have $\varnothing \equiv_T Y_0 \leq_T X_0 \leq_T Y_1 \leq_T X_1 \leq_T \cdots$. Since $\mathcal{A}$ is computably categorical relative to each X_i, and not computably categorical relative to each Y_i, we get

$$\varnothing \equiv_T Y_0 <_T X_0 <_T Y_1 <_T X_1 <_T \cdots .$$

In Lemma 2.7, we explain why $\mathcal{A}$ is relatively computably categorical relative to $\mathbf{0}'$.

Strategy for meeting $\mathcal{R}_i^k$ in isolation: Let $\mathcal{B} = \mathcal{B}_k = \Psi_k^{X_k}$. We take the following actions:

1. Choose a new large number ℓ and create two new root vertices a_1 and a_2 in $\mathcal{A}$, and b_1 and b_2 in $\mathcal{B}$.

2. Attach a loop of length 2 to a_1 and a_2 in $\mathcal{A}$, and to b_1 and b_2 in $\mathcal{B}$. Attach a loop of length $5\ell+1$ to a_1 in $\mathcal{A}$ and b_1 in $\mathcal{B}$; and attach a loop of length $5\ell+2$ to a_2 in $\mathcal{A}$ and b_2 in $\mathcal{B}$. The loop of length 2 is simply to identify a_1, a_2, b_1, and b_2 as root vertices.

$$\begin{array}{ll} a_1 : 2, 5\ell+1 & a_2 : 2, 5\ell+2 \\ b_1 : 2, 5\ell+1 & b_2 : 2, 5\ell+2 \end{array}$$

3. Wait for a stage s at which we see that $\Phi_i^{X_k} \colon \mathcal{A} \to \mathcal{B}$ maps $a_1 \mapsto b_1$ and $a_2 \mapsto b_2$. Let u be the use of this computation.

$$\begin{array}{l} a_1 : 2, 5\ell+1 \xrightarrow[\Phi]{} a_2 : 2, 5\ell+2 \\ b_1 : 2, 5\ell+1 \xrightarrow[\Phi]{} b_2 : 2, 5\ell+2 \end{array}$$

4. Choose a large number $v > u$. Attach loops of length $5\ell+3$ to a_1 in $\mathcal{A}$ and b_1 in $\mathcal{B}$. Attach loops of length $5\ell+4$ to a_2 in $\mathcal{A}$ and b_2 in $\mathcal{B}$ with use $X_k[s]\upharpoonright 2v+2$ where s is the current stage. *Restrain* $X_k\upharpoonright 2v+2$ so that it may not be changed by another requirement of a lower priority (to be clarified). Enumerate ℓ into $Y_{k'}^*$ for $k' \geq k$; this will be used to meet the $\mathcal{S}$ requirements.

$$\begin{array}{l} a_1 : 2, 5\ell+1, 5\ell+3 \xrightarrow[\Phi]{} a_2 : 2, 5\ell+2, 5\ell+4 \\ b_1 : 2, 5\ell+1, 5\ell+3 \xrightarrow[\Phi]{} b_2 : 2, 5\ell+2, 5\ell+4 \end{array}$$

5. Attach loops of length $5\ell+2$ to a_1 in $\mathcal{A}$ and b_1 in $\mathcal{B}$, and attach loops of length $5\ell+1$ to a_2 in $\mathcal{A}$ and b_2 in $\mathcal{B}$.

$$\begin{array}{l} a_1 : 2, 5\ell+1, 5\ell+2, 5\ell+3 \xrightarrow[\Phi]{} a_2 : 2, 5\ell+1, 5\ell+2, 5\ell+4 \\ b_1 : 2, 5\ell+1, 5\ell+2, 5\ell+3 \xrightarrow[\Phi]{} b_2 : 2, 5\ell+1, 5\ell+2, 5\ell+4 \end{array}$$

6. Enumerate v into X_k^*, which enumerates $2v+1$ into $X_k = Y_k \oplus X_k^*$, thus removing the loop of length $5\ell+3$ attached to b_1 and the loop of length $5\ell+4$ attached to b_2 in $\mathcal{B}$. Attach a loop of length $5\ell+4$ to b_1 in $\mathcal{B}$ and a loop of length $5\ell+3$ to b_2 in $\mathcal{B}$.

$$\begin{array}{l} a_1 : 2, 5\ell+1, 5\ell+2, 5\ell+3 \xrightarrow[\Phi]{} a_2 : 2, 5\ell+1, 5\ell+2, 5\ell+4 \\ b_1 : 2, 5\ell+1, 5\ell+2, 5\ell+4 \xrightarrow[\Phi]{} b_2 : 2, 5\ell+1, 5\ell+2, 5\ell+3 \end{array}$$

If $\Phi_i^{X_k}\colon \mathcal{A} \to \mathcal{B}_k$ is an isomorphism, it must be defined on a_1 and a_2 with some use u. In step (3), it must map $a_1 \mapsto b_1$ and $a_2 \mapsto b_2$ because a_1 and b_1 are the only elements with a loop of length $5\ell+1$, and a_2 and b_2 are the only elements with a loop of length $5\ell+2$. So at some stage s we see that $\Phi_i^{X_k[s]\upharpoonright u}$ maps $a_1 \mapsto b_1$ and $a_2 \mapsto b_2$. In steps (4), (5), and (6) we ensure that the elements have the following loops:

$$a_1 : 2, 5\ell+1, 5\ell+2, 5\ell+3 \xrightarrow[\Phi]{} a_2 : 2, 5\ell+1, 5\ell+2, 5\ell+4$$
$$b_1 : 2, 5\ell+1, 5\ell+2, 5\ell+4 \xrightarrow[\Phi]{} b_2 : 2, 5\ell+1, 5\ell+2, 5\ell+3$$

At stage (6), we enumerated v into X_k^*, but we still have $X_k \upharpoonright u = X_k[s] \upharpoonright u$. So $\Phi_i^{X_k}$ still maps $a_1 \mapsto b_1$ and $a_2 \mapsto b_2$, and this does not extend to an isomorphism.

Injury and restraint between different requirements $\mathcal{R}_i^k$: Recall that

$$Y_{i+1} = X_i \oplus Y_{i+1}^* \qquad \text{and} \qquad X_i = Y_i \oplus X_i^*.$$

Thus putting a restraint on X_k for some k means putting a restraint on each $X_{k'}^*$ and $Y_{k'}^*$, $k' \leq k$. Similarly, putting a restraint on Y_k for some k means putting a restraint on each $X_{k'}^*$, $k' < k$, and on each $Y_{k'}^*$, $k' \leq k$. The reader should keep this in mind throughout the proof.

If $\mathcal{R}_i^k$ finds a computation $\Phi_i^{X_k}$ in step (3) with use u, it restrains $X_k \upharpoonright u$. Another requirement $\mathcal{R}_{i'}^{k'}$ with $k' \leq k$ might have already chosen an element $v' < u$ in step (4), and want to enumerate v' into $X_{k'}^*$ in step (6). Since $k' \leq k$, this would enumerate an element into X_k as well, and potentially violating the restraint placed on X_k by $\mathcal{R}_i^k$. Similarly, enumerating an element ℓ into Y_k^* in step (4) might violate restraints placed by lower priority requirements.

We use the standard priority method; higher priority requirements $\mathcal{R}_{i'}^{k'}$ are allowed to violate the restraint placed by $\mathcal{R}_i^k$, in which case we say that $\mathcal{R}_i^k$ is *injured*; and when $\mathcal{R}_i^k$ places a restraint, it injures all lower priority requirements $\mathcal{R}_{i'}^{k'}$ which then have to choose a value v' greater than the restraint placed by $\mathcal{R}_i^k$. Each requirement, unless it is injured (which will only happen finitely many times) only places finitely many restraints. Thus by the usual arguments, the restraints are never violated. (Note that the numbers ℓ and v, which might be enumerated into X_k^* and Y_k^* respectively, are chosen to be large, i.e., large enough not to violate the restraints of any higher priority requirements. Lower priority requirements may later place restraints which would be violated by enumerating ℓ and v, but these lower priority requirements would then be injured.)

When a requirement is injured, it homogenizes the elements a_1 and a_2 it has been working with, and also the elements b_1 and b_2:

$$a_1 : 2, 5\ell+1, 5\ell+2, 5\ell+3, 5\ell+4 \qquad a_2 : 2, 5\ell+1, 5\ell+2, 5\ell+3, 5\ell+4$$
$$b_1 : 2, 5\ell+1, 5\ell+2, 5\ell+3, 5\ell+4 \qquad b_2 : 2, 5\ell+1, 5\ell+2, 5\ell+3, 5\ell+4$$

This means that any way of matching up a_1 and a_2 with b_1 and b_2 can be extended to an isomorphism. It then creates new elements a_1, a_2, b_1, and b_2 to work with.

Strategy for meeting $\mathcal{S}_i^k$: The requirement $\mathcal{S}_i^k$ is responsible for building a Y_k-computable isomorphism $f = \Gamma^{Y_k}$ between $\mathcal{A}$ and $\mathcal{M}_i^{Y_k}$. To define f, we must look at pairs of root nodes

a_1, a_2 in $\mathcal{A}$ and decide on images for them in $\mathcal{M} = \mathcal{M}_i^{Y_k}$. Given a_1, a_2, let ℓ be such that each of a_1 and a_2 has a loop of length $5\ell+1$ or $5\ell+2$. We can Y_k-computably look for a pair of root nodes c_1, c_2 in $\mathcal{M}$ which also have such loops. Finally, identify the requirement $\mathcal{R}_j^{k'}$ which was responsible for a_1, a_2. We have three cases in each of which we act differently:

- If $k' \leq k$: In step (4) of meeting the requirement $\mathcal{R}_j^{k'}$, we enumerate ℓ into Y_k^*; so by checking whether $\ell \in Y_k^*$, we can determine whether $\mathcal{R}_j^{k'}$ reached step (4) while working with a_1 and a_2. If it did not reach step (4), then exactly one of a_1, a_2 has a loop of length $5\ell+1$, and the other has a loop of length $5\ell+2$; so we can map whichever of a_1, a_2 has a loop of length $5\ell+1$ to whichever of c_1, c_2 has a loop of the same size, and this will extend to an isomorphism. If $\mathcal{R}_j^{k'}$ did reach step four, then exactly one of a_1, a_2 has a loop of length $5\ell+3$, and the other has a loop of length $5\ell+4$, and we can again match a_1, a_2 up with c_1, c_2.

- If $k' > k$ and $\mathcal{R}_j^{k'}$ is of higher priority than $\mathcal{S}_i^k$: $\mathcal{S}_i^k$ can non-uniformly know whether or not $\mathcal{R}_j^{k'}$ ever reached step (4); the rest is similar to the previous case.

- If $k' > k$ and $\mathcal{R}_j^{k'}$ is of lower priority than $\mathcal{S}_i^k$: In this case Y_k does not know whether $\mathcal{R}_j^{k'}$ reached step (4). Without loss of generality, we may assume that a_1 and c_1 have loops of length $5\ell+1$ and a_2 and c_2 have loops of length $5\ell+2$.

  $$a_1 : 2, 5\ell+1 \qquad a_2 : 2, 5\ell+2$$
  $$c_1 : 2, 5\ell+1 \qquad c_2 : 2, 5\ell+2.$$

  Then have f map $a_1 \mapsto c_1$ and $a_2 \mapsto c_2$. If $\mathcal{R}_j^{k'}$ never reaches step (4), then this extends to an isomorphism. The issue might be that $\mathcal{R}_j^{k'}$ reaches step (4), but that $\mathcal{M}$ delays adding the loops from step (4) until $\mathcal{R}_j^{k'}$ has reached step (6), so that $\mathcal{A}$ looks like

  $$a_1 : 2, 5\ell+1, 5\ell+2, 5\ell+3 \qquad a_2 : 2, 5\ell+1, 5\ell+2, 5\ell+4.$$

  $\mathcal{M}_j$ can now add loops so that it looks like:

  $$c_1 : 2, 5\ell+1, 5\ell+2, 5\ell+4 \qquad c_2 : 2, 5\ell+1, 5\ell+2, 5\ell+3$$

  This would defeat the isomorphism f.

  The solution is to force $\mathcal{R}_i^k$ to wait for $\mathcal{M}_j$ to copy $\mathcal{A}$ after step (4) before preceding on to step (5). After step (4), $\mathcal{A}$ looks like

  $$a_1 : 2, 5\ell+1, 5\ell+3 \qquad a_2 : 2, 5\ell+2, 5\ell+4$$

  and this forces $\mathcal{M}_j$, if it wants to be isomorphic to $\mathcal{A}$, to add loops so that it looks like

  $$c_1 : 2, 5\ell+1, 5\ell+3 \qquad c_2 : 2, 5\ell+2, 5\ell+4.$$

  Thus f remains an isomorphism.

  There is a complication in that $\mathcal{M}$ is a Y_k-computable structure, and we must build $\mathcal{A}$ computably. So we work with the stage-by-stage approximation to $\mathcal{M}$ given by the

stage-by-stage approximation to Y_k, and whenever $\mathcal{R}_j^{k'}$ sees more loops added to $\mathcal{M}$ (according to the current value of Y_k), it places a restraint on Y_k. If this restraint is ever violated by a higher priority requirement, we homogenize a_1 and a_2, so that the map f still extends to an isomorphism. (Recall that putting a restraint on Y_k for some k means putting a restraint on each $X^*_{k'}$, $k' < k$, and on each $Y^*_{k'}$, $k' \leq k$.)

Of course, $\mathcal{M}$ does not have to copy $\mathcal{A}$, and there will certainly be some j for which $\mathcal{M} \not\cong \mathcal{A}$ and $\mathcal{R}_j^{k'}$ gets stuck after step (4). To solve this we use a standard pressing strategy: the requirements $\mathcal{R}_j^{k'}$ must guess at the outcomes of the higher priority requirements. While $\mathcal{R}_j^{k'}$ is waiting for $\mathcal{M}$ to catch up, it starts a new instance guessing that $\mathcal{M}$ is not isomorphic to $\mathcal{A}$; when $\mathcal{M}$ does catch up, this new instance is destroyed (and homogenized).

The action in the second case works even when $k' \leq k$, it is just that the first case also works as well. But the action in the third case does not work when $k' \leq k$, and it might be helpful to the reader, to aid in understanding the construction, to think about why this is true: In the infinitary outcome, the requirement $\mathcal{R}_j^{k'}$ must be able to restrain Y_k, while still enumerating an element into $X^*_{k'}$; if $k' \leq k$, then enumerating an element into $X^*_{k'}$ would also enumerate an element into Y_k.

Priorities and guesses: We put a priority ordering on the requirements. A requirement $\mathcal{S}_i^k$ knows the outcome of the higher priority $\mathcal{R}$ requirements when it builds its isomorphism between $\mathcal{M}_i^{Y_k}$ and $\mathcal{A}$, and an $\mathcal{R}$ requirement must guess at the outcomes of the higher priority $\mathcal{S}$ requirements. The $\mathcal{R}$ requirements are the only requirements that are active during the construction and which can enumerate elements into the X_k and Y_k; the $\mathcal{S}$ requirements must be taken into account by the $\mathcal{R}$ requirements (e.g. by enumerating elements into the Y_k, or waiting to see loops in a structure $\mathcal{M}_i^{Y_k}$), and the isomorphisms they ask for can be defined after the construction is finished.

Full strategy for meeting $\mathcal{R}_i^k$: For simplicity we write $\mathcal{B} = \mathcal{B}_k$. Let F be a subset of the higher priority $\mathcal{S}$ requirements. For each such F, $\mathcal{R}_i^k$ can have a module which works under the assumption that the requirements in F are exactly the higher priority $\mathcal{S}$ requirements that $\mathcal{R}_i^k$ needs to wait for. We call this module a module of $\mathcal{R}_i^k$ with guess F. At certain stages in the module for working for F, we will have to wait to see some loops show up in a structure $\mathcal{M}_j^{Y_{k'}}$ for some $\mathcal{S}_j^{k'} \in F$; while waiting, we will start up a module for a set $G \subsetneq F$, which itself might start up another module, and so on.

When a module for $\mathcal{R}_i^k$ with guess F starts a new module with guess $G \subsetneq F$, we say that the module with guess F is the *parent* of the module with guess G, and the module with guess G is the *child* of the module with guess F. The module for $\mathcal{R}_i^k$ with F consisting of all of the higher priority $\mathcal{S}$ requirements is called the base module.

The module for $\mathcal{R}_i^k$ with guess F acts as follows:

1. Choose a new large number ℓ greater than the restraints of all higher priority requirements and also greater than the restraint of the parent modules of this module, and its parent, and so on. Create two new root vertices a_1 and a_2 in $\mathcal{A}$, and b_1 and b_2 in $\mathcal{B}$.

   Attach a loop of length 2 to a_1 and a_2 in $\mathcal{A}$, and to b_1 and b_2 in $\mathcal{B}$. Attach a loop of length $5\ell + 1$ to a_1 in $\mathcal{A}$ and b_1 in $\mathcal{B}$; and attach a loop of length $5\ell + 2$ to a_2 in $\mathcal{A}$ and b_2 in $\mathcal{B}$.

2. For each requirement $\mathcal{S}_j^{k'}$ in F, with $k' < k$, wait until we see a pair of root nodes c_1 and c_2 in $\mathcal{M}_j^{Y_{k'}}$ such that c_1 has a loop of length 2 and of length $5\ell + 1$, and c_2 has a loop of length 2 and of length $5\ell + 2$. If we ever see such elements, $\mathcal{R}_i^k$ puts a restraint r greater than the use of the oracle $Y_{k'}$ for the computation witnessing this. We say that such an $\mathcal{S}_j^{k'}$ has *caught up*.

   If at any stage s we ever see any other elements connected to c_1 or c_2, $\mathcal{R}_i^k$ puts a restraint r greater than the use of the oracle $Y_{k'}$ for the computation witnessing this. (Just put this restraint once per structure $\mathcal{M}_j^{Y_{k'}}$.) We say that such an $\mathcal{S}_j^{k'}$ has been *killed*.

   Remark 2.1. We note that the strategy only cares about its current component (the elements a_1 and a_2 and the adjacent loops it is currently working with). Therefore, if $\mathcal{S}_j^{k'}$ has been declared killed then this status is *not* global, i.e., it is internal for this particular version of the $\mathcal{R}_i^k$-strategy. We could of course make this status global and give the restraint some global priority (say, the priority of $\mathcal{S}_j^{k'}$). However, this is not necessary. This is because the injury in the construction will be only finite. Thus, if some higher priority requirement changes the set below the use witnessing that $\mathcal{S}_j^{k'}$ has been killed, we simply initialise $\mathcal{R}_i^k$. In particular, for one fixed component the situation in which $\mathcal{S}_j^{k'}$ was killed and then resurrected and then killed again etc. is impossible.

   While waiting, let G be the set of higher priority $\mathcal{S}$ requirements which have caught up but which have not been killed. Start a module of $\mathcal{R}_i^k$ with guess G. If we ever find that a new requirement has caught up, or that one which had caught up has now been killed, then the module with guess G must be homogenized (as described below), we reset G to be the new, larger, set of requirements which have caught up but which have not been killed, and start a module of $\mathcal{R}_i^k$ whose guess is the new G.

   If we ever see that every requirement in F has either caught up or been killed, move on to the next step.

3. Wait for a stage s at which we see that $\Phi_i^{X_k} \colon \mathcal{A} \to \mathcal{B}$ maps $a_1 \mapsto b_1$ and $a_2 \mapsto b_2$. Let u be the use of this computation. Enumerate ℓ into Y_k^*, injuring all lower priority requirements.

   Choose a large number $v > u$ and set a restraint $r = v + 1$. Attach loops of length $5\ell + 3$ to a_1 in $\mathcal{A}$ and b_1 in $\mathcal{B}$. Attach loops of length $5\ell + 4$ to a_2 in $\mathcal{A}$ and b_2 in $\mathcal{B}$ with use $X_k[s]\!\upharpoonright 2v + 2$ where s is the current stage.

4. For each requirement $\mathcal{S}_j^{k'}$ in F, with $k' < k$, wait until we see loops of length $5\ell + 3$ and $5\ell + 4$ attached to c_1 and c_2 respectively. $\mathcal{R}_i^k$ puts a restraint r greater than the use of the oracle $Y_{k'}$ for the computation witnessing this, and we say that $\mathcal{S}_j^{k'}$ has caught up.

   Again, if at any stage s we ever see any other elements connected to c_1 or c_2, $\mathcal{R}_i^k$ puts a restraint r greater than the use of the oracle $Y_{k'}$ for the computation witnessing this, and we say that $\mathcal{S}_j^{k'}$ has been killed.

   While waiting, we start new modules of $\mathcal{R}_i^k$ with guesses $G \subsetneq F$ as in step (2).

If we ever see that every requirement in F has either caught up or been killed, move on to the next step.

5. Attach loops of length $5\ell+2$ to a_1 in $\mathcal{A}$ and b_1 in $\mathcal{B}$, and attach loops of length $5\ell+1$ to a_2 in $\mathcal{A}$ and b_2 in $\mathcal{B}$.

   Enumerate v into X_k^*, which enumerates $2v+1$ into $X_k = Y_k \oplus X_k^*$, thus removing the loop of length $5\ell+3$ attached to b_1 and the loop of length $5\ell+4$ attached to b_2 in $\mathcal{B}$. Attach a loop of length $5\ell+4$ to b_1 in $\mathcal{B}$ and a loop of length $5\ell+3$ to b_2 in $\mathcal{B}$.

For $k' \neq k$, whenever we do anything in $\mathcal{A}$ (e.g. creating the elements a_1 and a_2 or adding loops to them) do the same in $\mathcal{B}_{k'}$ with no use.

Whenever a requirement increases its restraint, or enumerates an element, it injures all lower priority requirements. Each module of an injured requirement undergoes the homogenization procedure described below, and then the requirement restarts with just the base module at the first step.

When a module is to be homogenized, we do the following: for any loop on a_1 for which there is no corresponding loop on a_2, we add a loop of that length to a_2, and vice versa. So a_1 and a_2 have loops of exactly the same lengths attached to them. Do the same for b_1 and b_2.

Construction.

Recall that the $\mathcal{S}$ requirements do not take any action during the construction; we define the required isomorphisms after the construction.

At stage s, the first s $\mathcal{R}$-requirements are allowed to act. For each of these requirements, in order from highest priority to lowest priority, do the following: First, if the requirement has never before acted or was injured, start the base module of the requirement with F being the set of all higher priority $\mathcal{S}$ requirements. Then, execute the base module until we end up at a step where we have to wait for a larger stage s. Then, if there is a child module, execute the child module until it has to wait, then any child of the child module, and so on.

Verification.

Lemma 2.2. *Each requirement $\mathcal{R}_i^k$ is injured only finitely many times.*

Proof. We argue on induction that each $\mathcal{R}_i^k$ can injure the lower priority $\mathcal{R}$ requirements only finitely many times. To do this we suppose that a requirement $\mathcal{R}_i^k$ is never injured after some stage, and show that it injures the lower priority requirements only finitely many times.

An $\mathcal{R}_i^k$ module with guess F injures the lower priority requirements only finitely many times:

- once each time we find that a new requirement has caught up or been killed in step (2);
- once in step (3);

- once each time we find that a new requirement has caught up or been killed in step (4);
- once in step (5).

Now we argue that there are only finitely many $\mathcal{R}_i^k$ modules that ever run. We begin with the base module, say with guess F_0, and it has only one child at a time, with guess $F_1 \subsetneq F_0$; and its child module can have one child module, and so on. So at any one time, we have a chain of child modules with guesses $F_0 \supsetneq F_1 \supsetneq F_2 \supsetneq \cdots \supsetneq F_n$, and so $n \leq |F_0|$. A module can only homogenize its child module and start a new child module within the same step if a requirement has been found to have caught up or been killed, each of which can only happen once per requirement. The one exception to this is if the child module is homogenized in step (2) and the new child is started in step (4), but this can only happen once per module. So each module can have only finitely many child modules, each of which can have only finitely many child modules, and so on, and the depth is bounded by $|F_0|$. Thus there are only ever finitely many $\mathcal{R}_i^k$ modules running during the construction. □

Lemma 2.3. *Suppose that a module for $\mathcal{R}_i^k$ with guess G is never homogenized (which also means that $\mathcal{R}_i^k$ is never injured). Let $\mathcal{S}_j^{k'}$ be a higher priority requirement and suppose that $\mathcal{M}_j^{Y_{k'}} \cong \mathcal{A}$. Then $\mathcal{S}_j^{k'} \in G$.*

Proof. Suppose to the contrary that $\mathcal{S}_j^{k'} \notin G$. Then, because $\mathcal{S}_j^{k'}$ is contained in the guess by the base module, there must be some module with guess F containing $\mathcal{S}_j^{k'}$, which has a child module G not containing $\mathcal{S}_j^{k'}$, and neither module is ever homogenized.

So after some point the module with guess F must be stuck waiting at either stage (2) or (4), with G being exactly the set of higher priority requirements which have caught up but not been killed. But we will argue that since $\mathcal{M}_j^{Y_{k'}} \cong \mathcal{A}$, it must eventually catch up, and it can never be killed. This would cause the module with guess G to be homogenized, contradicting our initial assumption.

First, suppose that it is killed. Then at some stage s, we see in $\mathcal{M}_j^{Y_{k'}}[s]$ that there is a root vertex such that no root vertex in $\mathcal{A}$ has loops of the same lengths; and we put a restraint on the use so that $\mathcal{M}_j^{Y_{k'}}$ has such a vertex. (The restraint is never violated, or else $\mathcal{R}_i^k$ would be injured.) Thus we would have ensured that $\mathcal{M}_j^{Y_{k'}} \ncong \mathcal{A}$, which is not the case.

It must also catch up, because $\mathcal{M}_j^{Y_{k'}} \cong \mathcal{A}$, and so whenever a vertex shows up in $\mathcal{A}$ with certain loops, it must show up in $\mathcal{M}_j^{Y_{k'}}$; and so for some s, it must show up in $\mathcal{M}_j^{Y_{k'}}[s]$. □

Lemma 2.4. *For each k, $\mathcal{A}$ is computably categorical relative to Y_k.*

Proof. We must show that each Y_k-computable structure copy of $\mathcal{A}$ is Y_k-computably isomorphic to $\mathcal{A}$, i.e. that each requirement $\mathcal{S}_i^k$ is satisfied. Suppose that $\mathcal{A} \cong \mathcal{M}_j^{Y_k}$. For simplicity, write $\mathcal{M} = \mathcal{M}_j^{Y_k}$. Let f be the Y_k-computable isomorphism defined in the strategy for meeting $\mathcal{S}_j^k$. We argue that f is an isomorphism $\mathcal{M} \to \mathcal{A}$. It suffices to show that whenever we maps elements c_1, c_2 to a_1, a_2 respectively (as defined in the strategy for $\mathcal{S}_i^k$), c_1 and a_1 have the same lengths of loops attached to them, and c_2 and a_2 have the same lengths of loops attached to them.

If the module that built a_1, a_2 was ever homogenized, then any way of mapping c_1, c_2 to a_1, a_2 extends to an isomorphism. So suppose that the module that built a_1, a_2 was never homogenized. We have three cases from the definition of f:

1. If $k' \leq k$: If $\ell \notin Y_k^*$ then the module that built a_1, a_2 did not reach step (3), then a_1 is the unique root vertex in $\mathcal{A}$ (and c_1 is the unique root vertex in $\mathcal{M}$) with a loop of length $5\ell + 1$, and a_2 is the unique root vertex in $\mathcal{A}$ (and c_2 in $\mathcal{M}$) with a loop of length $5\ell + 2$. So mapping c_1, c_2 to a_1, a_2 respectively extends to an isomorphism.

   If $\ell \in Y_k^*$ then the module that built a_1, a_2 reached step (3), and a_1 is the unique root vertex in $\mathcal{A}$ (and c_1 is the unique root vertex in $\mathcal{M}$) with a loop of length $5\ell+3$, and a_2 is the unique root vertex in $\mathcal{A}$ (and c_2 in $\mathcal{M}$) with a loop of length $5\ell + 4$. So mapping c_1, c_2 to a_1, a_2 respectively extends to an isomorphism.

2. If $k' > k$ and $\mathcal{R}_j^{k'}$ is of higher priority than $\mathcal{S}_i^k$: Similar to above.

3. If $k' > k$ and $\mathcal{R}_j^{k'}$ is of lower priority than $\mathcal{S}_i^k$:

   Let F be the guess by the module of $\mathcal{R}_j^{k'}$ that built a_1, a_2. By Lemma 2.3, since $\mathcal{M}_i^{Y_k} \cong \mathcal{A}$, and the module with guess F is not homogenized, we have $\mathcal{S}_i^k \in F$.

   First of all, we argue that when we find a stage s and elements c_1 and c_2, such that in $\mathcal{M}[s]$ there is a loop of length $5\ell + 1$ on c_1 and of length $5\ell + 2$ on c_2, that in $\mathcal{M}[s]$ there is no loop of length $5\ell + 2$ on c_1 or $5\ell + 1$ on c_2. Otherwise, $\mathcal{S}_i^k$ would have been killed. (As in Lemma 2.3, if it is killed, then there is a restraint placed on Y_k which ensures that $\mathcal{M} \ncong \mathcal{A}$.)

   Then if the module with guess F does not make it to step (5), a_1 is the unique root vertex of $\mathcal{A}$ with a loop of length $5\ell + 1$, and a_2 is the unique root vertex with a loop of length $5\ell + 2$; so mapping c_1 to a_1 and c_2 to a_2 extends to an isomorphism.

   Then if the module with guess F does make it to step (5), a_1 is the unique root vertex of $\mathcal{A}$ with a loop of length $5\ell+3$, and a_2 is the unique root vertex with a loop of length $5\ell + 4$; we must argue that c_1 gets a loop of length $5\ell + 3$ and c_2 gets a loop of length $5\ell + 4$. In step (2) we wait to see c_1 get a loop of length $5\ell + 1$ and c_2 get a loop of length $5\ell + 2$; and then in step (4) we wait to see c_1 get a loop of length $5\ell + 3$ and c_2 a loop of length $5\ell + 4$. We put a new restraint on Y_k every time we see a new loop. Moreover, if c_2 got the loop of length $5\ell + 3$, or c_1 got the loop of length $5\ell + 4$, then $\mathcal{S}_i^k$ would be killed. So it must be c_1 that gets the loop of length $5\ell + 3$ and c_2 that gets the loop of length $5\ell + 4$.

This completes the proof of the claim. □

Lemma 2.5. *For each k, $\mathcal{B}_k$ is isomorphic to $\mathcal{A}$.*

Proof. The structure $\mathcal{A}$ consists entirely of pairs of root nodes a_1, a_2 produced by a module of an $\mathcal{R}$ requirement, and the elements forming the loops attached to these root nodes. Whenever we add elements a_1, a_2 to $\mathcal{A}$ for a requirement $\mathcal{R}_i^{k'}$, $k' \neq k$, we add the same sort of elements to $\mathcal{B}_k$ (i.e., $\mathcal{B}_k$ just copies $\mathcal{A}$). When we add elements a_1, a_2 to $\mathcal{A}$ for a requirement $\mathcal{R}_i^k$, we add elements b_1, b_2 to $\mathcal{B}_k$, and either a_1 has the same size loops as b_1

(and a_2 as b_2), or a_1 has the same size loops as b_2 (and a_2 and b_1); which case we are in depends on whether the requirement made it to step (5) or not. □

Lemma 2.6. *For each k, $\mathcal{B}_k$ is not X_k-computably isomorphic to $\mathcal{A}$.*

Proof. We must argue that each requirement $\mathcal{R}_i^k$ is satisfied, so that no X_k-computable map $\Phi_i^{X_k}$ is an isomorphism between $\mathcal{B}_k$ and $\mathcal{A}$. Fix some requirement $\mathcal{R}_i^k$ which we will show is satisfied. After some stage, it is no longer injured.

There is some module of $\mathcal{R}_i^k$, say with guess F, which is never homogenized, and which either waits forever in step (3) or reaches step (5). (The other options for a particular module are that it waits forever in step (2) or step (4), and in each of these cases it has a child module; we know from the arguments in the previous lemma that each module must have some last child module, and that the depth of child modules is bounded, so that there must be some module with no child module.)

First suppose that the module waits forever in step (3). The either $\Phi_i^{X_k}$ is partial, or it maps a_1 to some element other than b_1. If it is is partial then it obviously cannot be an isomorphism. If it maps a_1 to some element other than b_1, then it cannot extend to an isomorphism, as a_1 and b_1 each have a loop of length $5\ell + 1$, and no other elements of $\mathcal{A}$ or $\mathcal{B}$ do, as no other requirement or module has the same value of ℓ.

Now suppose that the module waits forever in step (5). Since the module passed through step (3), we have that $\Phi_i^{X_k}$ maps $a_1 \mapsto b_1$ and $a_2 \mapsto b_2$. (The requirement $\mathcal{R}_i^k$ puts a restraint on the use of the computation found in step (3), which cannot be violated by any lower priority requirement; and if a higher priority requirement violated the restraint, $\mathcal{R}_i^k$ would have been injured.) But we ensure in step (5) that a_1 has a loop of length $5\ell + 3$ and that b_1 does not, so that $\Phi_i^{X_k}$ does not extend to an isomorphism. □

Lemma 2.7. *$\mathcal{A}$ is computably categorical relative to $\mathbf{0}'$.*

Proof. Recall that the structure $\mathcal{A}$ is a directed graph consisting of infinitely many finite connected components. Each component consists of either two, three, four, or five cycles sharing a single vertex, the root vertex of the component. The root vertices can be identified as the only vertices of degree greater than two. Moreover, one can see from the construction that if two components are not isomorphic, it is because each of them contains a loop of a size that the other does not. (No component is a substructure of any other component.)

Thus $\mathcal{A}$ has a Σ_1 Scott family; the automorphism type of a component is determined by the existential formula saying which sizes of loops it contains. This Scott family is $\mathbf{0}'$-computable because each component is finite. As $\mathcal{A}$ has a $\mathbf{0}'$-computable Σ_1 Scott family, it is computably categorical relative to $\mathbf{0}'$. □

3 Finite computable dimension, and open questions

For each $n \in \omega - \{0\}$, Goncharov [Gon80b] constructed a computable structure which has computable dimension n. McCoy [McC02] used a forcing argument to observe that finite computable dimension does not relativize meaning that if $\mathcal{A}$ has computable dimension $n > 1$, with $n \in \omega$, then there is some oracle X relative to which $\mathcal{A}$ has dimension $\neq n$. Analysis of the construction shows that $X = \varnothing''$ would suffice. We make an improvement showing that $X = \varnothing'$ suffices.

Proposition 3.1. *If a computable structure has finite computable dimension > 1, then it has computable dimension ∞ relative to any degree above $\mathbf{0}'$.*

Proof. Let $\mathcal{A}$ be a computable structure. Relative to $\mathbf{0}'$, it is 1-decidable. By [DKLT13] $\mathcal{A}$ has a $\mathbf{0}'$-computable Σ^0_2 Scott family. If $\mathcal{B}$ is any other computable copy of $\mathcal{A}$ which is not computably isomorphic to $\mathcal{A}$, $\mathcal{A}$ and $\mathcal{B}$ are $\mathbf{0}''$-computably isomorphic. Thus $\mathcal{A}$ has computable dimension 1 or ∞ relative to $\mathbf{0}'$. If it has computable dimension 1 relative to $\mathbf{0}'$, then $\mathcal{A}$ and $\mathcal{B}$ are $\mathbf{0}'$-computably isomorphic, and so $\mathcal{A}$ has computable dimension ∞. □

We mention without proof the following result which is proven using a straightforward modification to the "special component" construction of Goncharov [Gon80b], as per Hirschfeldt [Hir00]. We believe that its proof would contain no new ideas, and would simply make the present paper 10 pages longer. We thus leave it as a strong conjecture.

Strong Conjecture 3.2. *There is a computable structure $\mathcal{A}$ of computable dimension 2 and a c.e. set L such that $\mathcal{A}$ has computable dimension 2 relative to L.*

A general question is then: What is the behaviour of the computable dimension of a computable structure after relativizing to various degrees? Some specific questions along these lines are as follows:

Question 3.3. For which degrees $\mathbf{d}$ is there a computable structure of computable dimension 2 which has computable dimension 2 relative to $\mathbf{d}$?

Question 3.4. For all $\mathcal{A}$ of computable dimension $1 < n < \infty$ is there X with $\varnothing <_T X <_T \varnothing'$ such that $\mathcal{A}$ does not have dimension n relative to X.

Question 3.5. Is there a degree $\mathbf{d} > \mathbf{0}$ and a computable structure of computable dimension 2 which has computable dimension 3 relative to $\mathbf{d}$?

Note that the reverse—having computable dimension 3 and then computable dimension 2 relative to $\mathbf{d}$—is less likely to happen:

Proposition 3.6. *Let $\mathcal{A}$ be a computable structure with computable dimension n, and with computable dimension m relative to $\mathbf{d}$, with $\mathbf{0}' \geq \mathbf{d} > \mathbf{0}$. Then $n \leq m$.*

Note that $\mathbf{d}$ also cannot be above $\mathbf{0}'$ by Proposition 3.1.

Proof. If $n > m$, then there are two non-isomorphic computable copies of $\mathcal{A}$ which are $\mathbf{d} \leq \mathbf{0}'$-computably isomorphic, and so $\mathcal{A}$ has computable dimension 1 or ∞, a contradiction. □

Question 3.7. For which degrees $\mathbf{d}$ is there a computably categorical structure with finite computable dimension 2 relative to $\mathbf{d}$? E.g. what about $\mathbf{d} = \mathbf{0}'$ or $\mathbf{d} = \mathbf{0}''$?

And going back to computable categoricity, we ask whether there is a degree $\mathbf{d}$ which is low in the sense that it is unable to make a computably categorical structure not computably categorical:

Question 3.8. Is there a degree $\mathbf{d} > \mathbf{0}$ such that, for all computably categorical $\mathcal{A}$, $\mathcal{A}$ is computably categorical relative to $\mathbf{d}$?

A construction can be used to show that no such X can be c.e.

References

[AK00] C. J. 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.

[AKMS89] Chris Ash, Julia Knight, Mark Manasse, and Theodore Slaman. Generic copies of countable structures. *Ann. Pure Appl. Logic*, 42(3):195–205, 1989.

[Chi90] John Chisholm. Effective model theory vs. recursive model theory. *J. Symbolic Logic*, 55(3):1168–1191, 1990.

[DKL$^+$15] Rodney G. Downey, Asher M. Kach, Steffen Lempp, Andrew E. M. Lewis-Pye, Antonio Montalbán, and Daniel D. Turetsky. The complexity of computable categoricity. *Adv. Math.*, 268:423–466, 2015.

[DKLT13] Rodney G. Downey, Asher M. Kach, Steffen Lempp, and Daniel D. Turetsky. Computable categoricity versus relative computable categoricity. *Fund. Math.*, 221(2):129–159, 2013.

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

[Gon77] S. S. Gončarov. The number of nonautoequivalent constructivizations. *Algebra i Logika*, 16(3):257–282, 377, 1977.

[Gon80a] S. S. Gončarov. Autostability of models and abelian groups. *Algebra i Logika*, 19(1):23–44, 132, 1980.

[Gon80b] S. S. Gončarov. The problem of the number of nonautoequivalent constructivizations. *Algebra i Logika*, 19(6):621–639, 745, 1980.

[Hir00] Denis R. Hirschfeldt. Degree spectra of relations on computable structures. *Bull. Symbolic Logic*, 6(2):197–212, 2000.

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

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

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

[McC02] Charles F. D. McCoy. Finite computable dimension does not relativize. *Arch. Math. Logic*, 41(4):309–320, 2002.

[MN18] Alexander G. Melnikov and Keng Meng Ng. Computable torsion abelian groups. *Adv. Math.*, 325:864–907, 2018.

[Rem81] J. B. Remmel. Recursively categorical linear orderings. *Proc. Amer. Math. Soc.*, 83(2):387–391, 1981.

[Sco65] Dana Scott. Logic with denumerably long formulas and finite strings of quantifiers. In *Theory of Models (Proc. 1963 Internat. Sympos. Berkeley)*, pages 329–341. North-Holland, Amsterdam, 1965.

[Smi81] Rick L. Smith. Two theorems on autostability in p-groups. In *Logic Year 1979–80 (Proc. Seminars and Conf. Math. Logic, Univ. Connecticut, Storrs, Conn., 1979/80)*, volume 859 of *Lecture Notes in Math.*, pages 302–311. Springer, Berlin-New York, 1981.

[Tur] Dan Turetsky. Coding in the automorphism group of a computably categorical structure. preprint.

[Ven92] Yu. G. Ventsov. The effective choice problem for relations and reducibilities in classes of constructive and positive models. *Algebra i Logika*, 31(2):101–118, 220, 1992.