The tree of tuples of a structure

Matthew Harrison-Trainor[‡] and Antonio Montalbán[§]

February 26, 2019

Abstract

Our main result is that there exist structures which cannot be computably recovered from their tree of tuples. This implies that there are structures with no computable copies which nevertheless cannot code any information in a natural/functorial way.

1 Introduction

Our main result is that there exist structures which cannot be computably recovered from their tree of tuples. As we will see below, the tree of tuples of a structure is the most natural labeled tree one can associate to a structure: it determines the structure up to isomorphism, it captures its back-and-forth structure, it codes all the information the structure codes in terms of families of families of ... of sets of numbers in a functorial way, and it is the top replicated labeled tree that can be effectively interpreted in the structure (top in terms of effective interpretability). Despite all of this, our result shows that some information is lost when considering this tree of tuples, even though this information is coded in a very subtle way that is hard to recover.

1.1 The information content of a structure

A common question in computable structure theory is, given a structure, what is it information content? The question can be interpreted in various ways, depending on how we represent information and how we retrieve content. The first approach is to represent information by subsets of ω and retrieve it by enumerating it computably:

Definition 1.1. A set $X \subseteq \omega$ is *c.e.-coded* in a structure $\mathcal{A}$ if X is computably enumerable in the atomic diagram $D(\mathcal{A})$ of any presentation of $\mathcal{A}$.

The class of sets that are c.e.-coded in a structure is characterized by Knight's Theorem [Kni86]: A set X is c.e.-coded in a structure $\mathcal{A}$ if and only if X is enumeration reducible to the $\exists$-type of some tuple from $\mathcal{A}$, where the $\exists$-type of a tuple $\bar{a}$ in $\mathcal{A}$ is given by

$$\exists\text{-}tp_{\mathcal{A}}(\bar{a}) = \{\ulcorner\varphi\urcorner : \varphi(\bar{x}) \text{ is an } \exists\text{-formula such that } \mathcal{A} \models \varphi(\bar{a})\}.$$

[‡]The first author was partially supported by a Banting fellowship.

[§]The second author was partially supported by NSF grant DMS-1363310 and by the Simons fellowship.

If we ask about sets that are *uniformly c.e.-coded*, i.e., sets X for which there is a c.e.-operator W for which $X = W^{D(\widehat{\mathcal{A}})}$ for all copies $\widehat{\mathcal{A}}$ of $\mathcal{A}$, we get that there is a greatest enumeration degree that is uniformly c.e.-coded in a structure $\mathcal{A}$ is given by the $\exists$-theory of $\mathcal{A}$. This is all well-known and well-understood. Of course, there is more information in a structure than the sets its c.e.-codes; the next step up is to consider families of sets.

1.2 Families of sets

Families of subsets of ω are much richer objects than just sets. A large number of examples built in computable structure theory are structures that are defined out of families of sets.

Definition 1.2. An enumeration of a countable family $\mathcal{F} \subseteq \mathcal{P}(\omega)$ is a set $W \subseteq \omega^2$ such that $\mathcal{F} = \{W^{[i]} : i \in \omega\}$, where $W^{[i]}$ is the ith column of W. The order of the columns in W and their multiplicity is not relevant. To avoid issues with multiplicity, we may always assume that in an enumeration of a family each column is replicated infinitely many times.

A family $\mathcal{F}$ is *computably enumerable* in a set X if there is an X-c.e. enumeration of $\mathcal{F}$. A family $\mathcal{F}$ is *c.e.-coded* in a structure $\mathcal{A}$ if it is computably enumerable in the atomic diagrams of all copies of $\mathcal{A}$. Or, equivalently, if the set of X over which $\mathcal{F}$ is computably enumerable is Muchnik reducible to the degree spectrum of $\mathcal{A}$. $\mathcal{F}$ is *uniformly c.e.-coded* in a structure $\mathcal{A}$ if there is a c.e.-operator W such that $W^{D(\widehat{\mathcal{A}})}$ is an enumeration of $\mathcal{F}$ for all copies $\widehat{\mathcal{A}}$ of $\mathcal{A}$. Or, equivalently, if the set of X over which $\mathcal{F}$ is computably enumerable is Medvedev reducible to the degree spectrum of $\mathcal{A}$.

We would like to get a result like the one by Knight mentioned above for the case of families of sets. The second author [Mon13b] proved the following unsatisfactory version: Let $\{\Theta_e : e \in \omega\}$ be the standard computable enumeration of all enumeration operators. If a family $\mathcal{F}$ is uniformly c.e.-coded in a structure $\mathcal{A}$, then there exist a uniformly computable list of Σ^{c}_3 formulas $\varphi_\ell(\bar{y})$ such that

$$\mathcal{F} = \{\Theta_\ell(\exists\text{-tp}_{\mathcal{A}}(\bar{b})) : \ell \in \omega,\ \bar{b} \in A^{<\omega},\ \mathcal{A} \models \varphi_\ell(\bar{b})\}. \tag{$\circledast$}$$

This is not an if-and-only-if condition, and the use of Σ^{c}_3 formulas is clearly undesirable. However, if we strengthen the way the family is coded by the structure, then we can actually require the formulas be Σ^{c}_1:

Definition 1.3. A family $\mathcal{F}$ is *functorially c.e.-coded* if also there is a computable operator that, given two copies $\widehat{\mathcal{A}}$ and $\widetilde{\mathcal{A}}$ of $\mathcal{A}$ and an isomorphism f between them, produces a permutation of ω matching the columns of $W^{D(\widehat{\mathcal{A}})}$ and $W^{D(\widetilde{\mathcal{A}})}$ in a *functorial way*, meaning that this operator maps the identity to the identity and preserves composition of isomorphisms. See Definition 2.1.

Results from [HTMMM17] (see Section 2 below) show that a family is functorially c.e.-coded if and only if it is effectively interpreted in $\mathcal{A}$, and in and only if it is Σ-definable in $\mathcal{A}$ without parameters. In particular, we get the following characterization.

Proposition 1.4. *A family $\mathcal{F}$ is functorially c.e.-coded in a structure $\mathcal{A}$ if and only if there exists a uniformly computable list of Σ^{c}_1 formulas $\varphi_\ell(\bar{x}, \bar{y})$ such that ($\circledast$) holds.*

We prove this as Proposition 3.1 below.

It follows that Montalbán's necessary condition (⋇) for families that are uniformly c.e.-coded in a structure is not true if we require that the formulas φ_ℓ are Σ^{c}_1: In that case we would have that $\mathcal{F}$ is effectively interpretable in $\mathcal{A}$ (see Section 2), and Kalimullin and Puzarenko [Kal09, KP09, Kal12] showed that there exists a family $\mathcal{F}$ and a structure $\mathcal{A}$ such that $\mathcal{F}$ is uniformly c.e.-coded in $\mathcal{A}$ (they called this *strongly reducible*) but not effectively interpretable in $\mathcal{A}$ even with parameters (they called this Σ-definable).

We say that a family $\mathcal{F}$ is *uniformly c.e.-coded* in another family $\mathcal{S}$ if there is a c.e.-operator W such that, given an enumeration V of $\mathcal{S}$, produces an enumeration W^V of $\mathcal{F}$. We say that $\mathcal{F}$ is *functorially reducible* to $\mathcal{S}$ if there is also a computable operator that, given two enumerations $\hat{V}$ and $\tilde{V}$ of $\mathcal{S}$ and a matching permutation between them, outputs a matching permutation between $W^{\hat{V}}$ and $W^{\tilde{V}}$ in a functorial way.

Functorially c.e.-coding seems to be a structurally better notion than uniformly c.e.-coding. It follows from Proposition 1.4 that, among all the families that are functorially c.e.-coded in a structure $\mathcal{A}$, there is a best one—one to which all the others can all be functorially reduced—namely the family of $\exists$-types of all the tuples in $\mathcal{A}$. It will follow from our main result, Theorem 1.10, that there is a structure $\mathcal{A}$ with no computable copies, for which the family of $\exists$-types has a computable enumeration (using the observation that the family of $\exists$-types of a structure is functorially reducible to the tree of tuples of that structure). We then get the following corollary.

Corollary 1.5. *There is a structure $\mathcal{A}$ which has no computable presentations, but so that all families that are functorially c.e.-coded in $\mathcal{A}$ are computable.*

1.3 Families of families of sets

We could take an extra step and consider countable families of families of subsets of ω. Furthermore, we can consider n-families, where an $(n+1)$-*family* is a family of n-families, and 0-family is a number. We get the same story: The second author [Mon13b] got a formula like the one in (⋇) satisfied by all n-families c.e.-coded in a structure, now using $\Sigma^{\mathsf{c}}_{2n-1}$ formulas. We already know from the case $n = 2$ that this formula does not give an if-and-only-if condition. We will prove that if we extend the notion of functorially c.e.-coded, then we do get a nice characterization of the n-families that are functorially c.e.-coded on a structure, and that there is a top one.

Why stop at finite levels? The same works for α-families for any computable ordinal α. An α-family is a family that contains β-families for $\beta < \alpha$. They can be represented by labeled trees, where each node is tagged by the an ordinal expressing which type of family it is representing, and each leaf—representing a 0-family—is labeled with a natural number. To avoid multiplicity issues, we will require that each branch is replicated infinitely often. We call these trees *replicated labeled α-trees*. A replicated labeled α-tree is *Medvedev reducible* (the analog of uniformly c.e.-coded) in a structure $\mathcal{A}$ if there is a c.e.-operator such that $W^{\widehat{\mathcal{A}}}$ is a presentation for the tree for each copy $\widehat{\mathcal{A}}$ of $\mathcal{A}$. A replicated labeled α-tree is *functorially reducible* to a structure $\mathcal{A}$ if also there is a computable operator that, given two copies $\widehat{\mathcal{A}}$ and $\widetilde{\mathcal{A}}$ of $\mathcal{A}$ and an isomorphism f between them, produces an isomorphism of the trees $W^{\widehat{\mathcal{A}}}$ and $W^{\widetilde{\mathcal{A}}}$ in a functorial way. See Section 2.

For each structure $\mathcal{A}$, there is a replicated labeled α-tree $\mathcal{T}_\alpha(\mathcal{A})$ that is naturally associated to it. It consist of the finite sequences of the form

$$\langle(\bar{a}_0, \beta_0, n_0), (\bar{a}_1, \beta_1, n_1), ...(\bar{a}_k, \beta_k, n_k)\rangle,$$

where $\bar{a}_0, \bar{a}_1, ..., \bar{a}_k$ are tuples form $\mathcal{A}$, the β_i's are ordinals satisfying $\alpha > \beta_0 > \beta_1 > \cdots > \beta_k$, and the n_i's are numbers used just to replicate each branch infinitely often. Such node is tagged by β_k. The leaves of the trees are labeled by a number coding the finite atomic diagram of the tuple $\bar{a}_0\bar{a}_1...\bar{a}_k$ in $\mathcal{A}$. Recall that the *finite atomic diagram* of a tuple $\bar{a}$, denoted $D_\mathcal{A}(\bar{a})$ is a finite binary string recording the truth values of the atomic formulas about $\bar{a}$ in $\mathcal{A}$ that use the first $|\bar{a}|$ relation symbols in the vocabulary, viewing all symbols of the vocabulary as relations. See [MonP1, Chapter I].

Theorem 1.6. *Given a structure $\mathcal{A}$, $\mathcal{T}_\alpha(\mathcal{A})$ is functorially reducible to $\mathcal{A}$ and any other replicated labeled α-tree that is functorially reducible to $\mathcal{A}$ is functorially reducible to $\mathcal{T}_\alpha(\mathcal{A})$.*

See Theorem 3.3.

1.4 Replicated labeled trees

One can take yet another step further and consider families of families of families of ... going on forever—in a sense ω^*-families. These are represented by trees, but now we do not require well-foundedness and we omit the ordinal tags on the nodes. Since we might have no leaves, we put natural number labels on all nodes of the tree. Again, to avoid issues with multiplicity, we require each branch to be repeated infinitely often.

Definition 1.7. A *replicated labeled tree* consist of a tree T, which has a parent function, together with a labeling function $\ell\colon T \to \omega$, and satisfies that for every $\sigma \in T$ with parent τ, there exist infinitely many children $\tilde{\sigma}$ of τ such that $\mathcal{T}_{\tilde{\sigma}}$ is isomorphic to T_σ. We use $\mathbb{RLT}$ to denote the class of replicated labeled trees.

It is not hard to see how a family of replicated labeled trees can be naturally represented by a single replicated labeled tree. Thus, replicated labeled trees encapsulate all the objects we have seen so far: sets, families of sets, and even α-families of sets. To each structure $\mathcal{A}$, there is a replicated labeled tree that is naturally associated to it:

Definition 1.8. The *tree of tuples* of a structure $\mathcal{A}$, $\mathcal{T}(\mathcal{A})$, consist of all the tuples from $\mathcal{A}$ ordered by inclusion where each tuple $\bar{a}$ is labeled by a number coding its finite atomic diagram $D_\mathcal{A}(\bar{a})$. To make it a replicated labeled tree, we define $\mathcal{T}_\infty(\mathcal{A})$ by replicating each branch infinitely often.

The tree $\mathcal{T}(\mathcal{A})$ appeared in Friedman–Stanely's [FS89a] proof that trees are Borel complete; the Borel reduction was $\mathcal{A} \mapsto \mathcal{T}(\mathcal{A})$. The tree $\mathcal{T}_\infty(\mathcal{A})$ appeared in their proof that linear orderings are Borel complete; this proof can be split into two steps, first mapping $\mathcal{A}$ to $\mathcal{T}_\infty(\mathcal{A})$ and then coding $\mathcal{T}_\infty(\mathcal{A})$ into a linear order (see Section 7). That is, as opposed to the case of α-trees, we now have that the structure $\mathcal{A}$ can be uniquely determined from $\mathcal{T}(\mathcal{A})$: Given two structures $\mathcal{A}$ and $\mathcal{B}$,

$$\mathcal{A} \cong \mathcal{B} \iff \mathcal{T}(\mathcal{A}) \cong \mathcal{T}(\mathcal{B}) \iff \mathcal{T}_\infty(\mathcal{A}) \cong \mathcal{T}_\infty(\mathcal{B}).$$

The proof of this is that the tree $\mathcal{T}(\mathcal{A})$ contains all of the back-and-forth information from $\mathcal{A}$; so knowing that $\mathcal{T}(\mathcal{A}) \cong \mathcal{T}(\mathcal{B})$ is enough to construct an isomorphism between $\mathcal{A}$ and $\mathcal{B}$ using the standard back-and-forth argument.

To show that $\mathcal{T}_\infty(\mathcal{A})$ is the canonical replicated labeled tree associated to $\mathcal{A}$ we will show the following:

Theorem 1.9. *Given a structure $\mathcal{A}$, $\mathcal{T}_\infty(\mathcal{A})$ is functorially reducible to $\mathcal{A}$ and every other replicated labeled tree that is functorially reducible to $\mathcal{A}$ is functorially reducible to $\mathcal{T}_\infty(\mathcal{A})$.*

See Theorem 3.4.

When we said that $\mathcal{A}$ is determined from $\mathcal{T}_\infty(\mathcal{A})$, all we meant is that $\mathcal{A}$ is the only structure whose tree of tuples is isomorphic to $\mathcal{T}_\infty(\mathcal{A})$. But, how difficult is it to recover $\mathcal{A}$ from $\mathcal{T}_\infty(\mathcal{A})$? A path through $\mathcal{T}_\infty(\mathcal{A})$ corresponds to a sequence of tuples $\bar{a}_0 \subseteq \bar{a}_1 \subseteq \cdots$, or equivalently, to map $f\colon \omega \to \mathcal{A}$. If this map is onto, then the labels throughout this path code $\bigcup_n D_{\mathcal{A}}(\bar{a}_n)$, which is the atomic diagram of the congruence presentation $f^{-1}(\mathcal{A})$ given by the pull-back of $\mathcal{A}$ through f. (It is a congruence presentation because elements of $\mathcal{A}$ may be repeated, and hence the equality relation is interpreted by an equivalence relation.) Thus, if we could find an onto path, we could produce a copy of $\mathcal{A}$. The set of onto paths is co-meager, thus, in the sense of category, almost all paths code $\mathcal{A}$. Thus, every copy $\mathcal{S}$ of $\mathcal{T}_\infty(\mathcal{A})$ can compute a copy of $\mathcal{A}$ with the help of a generic enough oracle. However, lots of genericity might be needed.

Our main theorem is that, for some structures, $\mathcal{T}(\mathcal{A})$ cannot compute $\mathcal{A}$ back.

Theorem 1.10. *There is a structure $\mathcal{A}$ with no computable copies but for which $\mathcal{T}(\mathcal{A})$ has a computable copy.*

We will prove this theorem in Section 5. In Section 6 we strengthen this.

Corollary 1.11. *For each computable ordinal α, there is a structure $\mathcal{A}$ with no Δ^0_α-computable copies but for which $\mathcal{T}(\mathcal{A})$ has a computable copy.*

Knight, Soskova, and Vatev have informed the authors that they have independently shown that there are no $\mathcal{L}_{\omega_1\omega}$ formulas that uniformly (in the choice of signature) interpret a structure $\mathcal{A}$ in the tree $\mathcal{T}(\mathcal{A})$; see [KSV]. This follows from the relativization of Corollary 1.11 as if $\mathcal{A}$ is interpreted in $\mathcal{T}(\mathcal{A})$ using Σ^X_α formulas, then any copy of $\mathcal{T}(\mathcal{A})$ $\Delta^{0,X}_\alpha$-computes a copy of $\mathcal{A}$.

1.5 Degree spectra of trees

Informally, we say that a class $\mathcal{C}$ of structures is universal if for any structure of any signature, there is a structure is $\mathcal{C}$ that has the same computability-theoretic properties such as the same degree spectrum, the same computable dimension, etc. More formally, we can ask that the two structures be effectively bi-interpretable, which implies that they have the same computability-theoretic properties. Many classes of structures—graphs, groups, rings, and fields—have been shown to be universal. Other classes of structures—linear orders, boolean algebras, torsion-free abelian groups, real closed fields—are known not to be universal. Usually in these cases, there is some specific computability-theoretic property that

cannot be realized in the class. For example, if a boolean algebra has a low copy, then it has a computable copy, and so there is no boolean algebra with Slaman-Wehner degree spectrum.

The case of trees is different however. Trees are not universal in the sense that there are structures which are not bi-interpretable with any tree; for example, two structure which are bi-interpretable have the same automorphism group, but no tree has automorphism group $\mathbb{Z}$. But we do not know of any purely computability-theoretic property that can be realized in some structure, but not in a tree. This is because as discussed previously trees can code sets, families, families of families, and so on, and so any example that is constructed in such a way can be replicated by a tree. So for example there is a tree with each possible computable dimension, there is a tree with Slaman-Wehner degree spectrum, and there is a tree which is computably categorical but not relatively computably categorical.

This is an undesirable situation, because what we really care about is the informal notion of universality: is it true that for every structure of any signature, there is a tree with the same computability-theoretic properties? The formal notion involving bi-interpretability is just an attempt to capture this informal notion. So when we show that a class of structures is not universal, we really want to exhibit some particular computability-theoretic property that cannot be realized in that class.

For trees, the best approach seems to be by looking at degree spectra.

Question 1.12. Is every degree spectrum the degree spectrum of a tree?

What we are looking for is a way of transforming a structure into a tree, and the tree of tuples is the obvious thing to try. (Technically, the tree of tuples also has unary label relations; but one can construct a tree in which these labels are coded directly into the tree.) If $\mathcal{A}$ and $\mathcal{T}(\mathcal{A})$ had the same degree spectrum, then this question would have a positive answer.

But of course Theorem 1.10 says exactly the opposite, and so this attempt to answer the question fails. And as the tree of tuples seems like the best transformation of a structure into a tree that one could hope for, we conjecture that the answer is negative: there is a degree spectrum which is not the degree spectrum of a tree. Answering this question would tell use something new and very interesting about degree spectra.

2 Effective interpretability and functorial reductions

A commonly used reducibility between structures is Medvedev reducibility: A structure $\mathcal{A}$ is *Medvedev reducible* to a structure $\mathcal{B}$ if there is a Turing operator Φ such that, when given the atomic diagram $D(\widehat{\mathcal{A}})$ of a copy of $\mathcal{A}$ as oracle, it produces the atomic diagram of a copy of $\mathcal{B}$. However, we know of no structural characterization for Medvedev reducibility. Recently, Harrison-Trainor, Melnikov, Miller, and Montalbán [HTMMM17] showed that a strengthening of the notion of Medvedev reducibility called functorial reducibility [MPSS18], can be characterized syntactically using the notion of effective interpretability.

Let us first define functorial reducibility:

Definition 2.1. [MPSS18, Definition 3.1] Given structures $\mathcal{A}$ and $\mathcal{B}$, a *computable functor* F from $\mathcal{A}$ to $\mathcal{B}$ consists of two computable operators Φ and Φ_* such that

1. for every copy $\widehat{\mathcal{A}}$ of $\mathcal{A}$, $\Phi^{D(\widehat{\mathcal{A}})}$ is the atomic diagram of a structure $F(\mathcal{A})$ isomorphic to $\mathcal{B}$;

2. for every isomorphism $f : \widehat{\mathcal{A}} \to \widetilde{\mathcal{A}}$ between structures isomorphic to $\mathcal{A}$, $\Phi_*^{D(\widehat{\mathcal{A}}) \oplus f \oplus D(\widetilde{\mathcal{A}})}$ is an isomorphism $F(f)\colon F(\widehat{\mathcal{A}}) \to F(\widetilde{\mathcal{A}})$;

3. F is *functorial*, i.e.,

   (a) $F(\mathrm{id}_{\widehat{\mathcal{A}}}) = \mathrm{id}_{F(\widehat{\mathcal{A}})}$ (where $\mathrm{id}_{\widehat{\mathcal{A}}}$ and $\mathrm{id}_{F(\widehat{\mathcal{A}})}$ are the identities on $\widehat{\mathcal{A}}$ and $F(\widehat{\mathcal{A}})$ respectively), and

   (b) $F(f \circ g) = F(f) \circ F(g)$ for all isomorphisms $f\colon \widehat{\mathcal{A}} \to \widetilde{\mathcal{A}}$ and $g\colon \widetilde{\mathcal{A}} \to \check{\mathcal{A}}$.

Notice that Φ itself is simply a Medvedev reduction from $\mathcal{A}$ to $\mathcal{B}$.

On the syntactic side we have effective interpretations, which were first introduced by Montalbán [Mon13a] (see also [HTMMM17]), though they are equivalent to the parameterless version of the well-studied notion of Σ-definability (see Ershov [Ers96]). Effective interpretations are like interpretations in model theory with a few differences: The elementary first-order definitions of a model-theoretic interpretation are now replaced by effective Δ_1^{c} definitions, and the interpretation is allowed to use tuples of arbitrary sizes. Recall that a relation on a structure is Δ_1^{c}-definable if and only if it is uniformly relatively intrinsically computable.

Definition 2.2. An *effective interpretation* of $\mathcal{A} = (A, P_0^{\mathcal{A}}, P_1^{\mathcal{A}}, \ldots)$ in $\mathcal{B}$ consists of:

- a Δ_1^{c}-definable subset $Dom_{\mathcal{A}}^{\mathcal{B}} \subseteq \mathcal{B}^{<\omega}$,
- a Δ_1^{c}-definable equivalence relation $\sim$ on $Dom_{\mathcal{A}}^{\mathcal{B}}$,
- a sequence of uniformly Δ_1^{c}-definable sets $R_i \subseteq (Dom_{\mathcal{A}}^{\mathcal{B}})^k$, where k is the arity of P_i, which respect $\sim$,
- a surjective map $f_{\mathcal{A}}^{\mathcal{B}}\colon Dom_{\mathcal{A}}^{\mathcal{B}} \to A$ which induces an isomorphism

$$f_{\mathcal{A}}^{\mathcal{B}}\colon (Dom_{\mathcal{A}}^{\mathcal{B}}/\sim; R_0/\sim, R_1/\sim, \ldots) \to \mathcal{A}.$$

It is not hard to see that effective interpretations produce computable functors. Harrison-Trainor, Melnikov, Miller, and Montalbán [HTMMM17] showed one can go the other way, and produce an effective interpretation out of a computable functor.

Two structures are said to be Σ-*equivalent* if they are Σ-definable in each other. (Recall that Σ-definability is the same as effective interpretability but allowing the use of a finite tuple of parameters.) These notions have been widely studied by the Russian school of computable structure theory. It will follow from our results that there is a structure that is not Σ-equivalent to any labeled tree.

A stronger notion of equivalence called *effectively bi-interpretable* was introduced by the second author in [Mon14], where the structure not only need to be effectively interpretable in each other, but also the composition of the isomorphisms $f_{\mathcal{A}}^{\mathcal{B}} \circ f_{\mathcal{B}}^{\mathcal{A}}$ and $f_{\mathcal{B}}^{\mathcal{A}} \circ f_{\mathcal{A}}^{\mathcal{B}}$ have to be Δ_1^{c}-definable in the respective structures. Showing that there is a structure that is not

effectively bi-interpretable to any labeled tree does not require using our main theorem: The linear order $(\mathbb{Z}; <)$ is not effectively bi-interpretable with any labeled tree because bi-interpretation preserve automorphism groups, and the automorphism group of any tree must contain an element of order two, unless it is rigid, because it must then have two branches that are isomorphic. On the other hand, the automorphism group of $(\mathbb{Z}; <)$ is $(\mathbb{Z}; +)$, which does not contain any element of order two.

3 The family of types

To each set, family of sets, family of families, etc. we can associate a structure that captures its information content so that we can apply the tools we already have to compare structures, as for instance the notions of functorial reducibility and effective interpretability. In this paper we represent families of sets by labeled trees as that allows us to easily generalize to α-families and ω^*-families. There are various other equivalent ways to represent families: bouquet graphs are standard in the west, while other structures are also common in Russia (see, for instance, [KP09]).

We associate to a set $S \subseteq \omega$ a labeled tree $\mathcal{T}_S$ of height one which contains a leaf labeled n if and only if $n \in S$. We associate to a family of sets $\mathcal{F}$ a labeled tree $\mathcal{T}_{\mathcal{F}}$ of height two as follows: For each set $S \in \mathcal{F}$ we we add infinitely many branches of the form $\mathcal{T}_S$. It is not hard to see that $\mathcal{F}$ is functorially c.e.-coded in a structure $\mathcal{A}$ if and only if $\mathcal{T}_{\mathcal{F}}$ is functorially reducible to $\mathcal{A}$.

Proposition 3.1. *A family of sets $\mathcal{F}$ is functorially c.e.-coded in a structure $\mathcal{A}$ if and only if there is a Σ^{c}_1 formula φ and a computable sequence of Σ^{c}_1 formulas ψ_n, $n \in \omega$, such that*[*]

$$\mathcal{F} = \{\{n \in \omega : \mathcal{A} \models \psi_n(\bar{a})\} : \bar{a} \in A^{<\omega}, \mathcal{A} \models \varphi(\bar{a})\}.$$

Proof. The right-to-left direction is straightforward, as we have direct way of producing an enumeration of $\mathcal{F}$ out of $\mathcal{A}$ in a functorial way.

For the left-to-right direction, consider an effective interpretation of $\mathcal{T}_{\mathcal{F}}$ within $\mathcal{A}$. Let $\psi(\bar{x})$ be the Σ^{c}_1 formula that says that $\bar{x}$ is in the domain of the interpretation of $\mathcal{T}_{\mathcal{F}}$ within $\mathcal{A}$, and that it represents a node of height one. Let $\varphi_n(\bar{x})$ be the formula that says that, in the interpretation of $\mathcal{T}_{\mathcal{F}}$ within $\mathcal{A}$, $\bar{x}$ has a child labeled n. □

Corollary 3.2. *If a family $\mathcal{F}$ is functorially c.e.-coded in a structure $\mathcal{A}$, it is functorially reducible to its family of $\exists$-types*

$$\{\{\ulcorner \phi \urcorner : \phi(\bar{x}) \text{ is an } \exists\text{-formula and } \mathcal{A} \models \phi(\bar{a})\} : \bar{a} \in A^{<\omega}\}.$$

The same idea works for α-families.

Theorem 3.3. *Let α be a computable ordinal. Suppose that an α-family $\mathcal{F}$ is functorially reducible to a structure $\mathcal{A}$. Then $\mathcal{F}$ is functorially reducible to $\mathcal{T}_\alpha(\mathcal{A})$.*

[*] We will often refer to a single Σ^{c}_1 formula $\varphi(\bar{x})$ where the number of variables is flexible, when we actually mean a computable sequence $\{\varphi_n(x_1, ..., x_n) : n \in \omega\}$ and interpret $\varphi(\bar{a})$ as $\varphi_{|\bar{a}|}(\bar{a})$.

Proof. Consider an effective interpretation of $\mathcal{T}_{\mathcal{F}}$ in $\mathcal{A}$ with domain $\mathcal{T}_{\mathcal{F}}^{\mathcal{A}} \subseteq A^{<\omega}$. We define an effective interpretation of $\mathcal{T}_{\mathcal{F}}$ in $\mathcal{T}_\alpha(\mathcal{A})$ whose domain will be a subset of $\mathcal{T}_\alpha(\mathcal{A})$. Put a tuple $\sigma = \langle(\bar{a}_0, \beta_0, n_0), (\bar{a}_1, \beta_1, n_1), ...(\bar{a}_k, \beta_k, n_k)\rangle \in \mathcal{T}_\alpha(\mathcal{A})$ in the domain of the new interpretation if each of $(n_i, \bar{a}_i)$ is in the domain of the interpretation with tag β_i and parent $(n_i, \bar{a}_{i-1})$. Of course, let $\langle(\bar{a}_0, \beta_0, n_0), (\bar{a}_1, \beta_1, n_1), ...(\bar{a}_{k-1}, \beta_{k-1}, n_{k-1})\rangle$ be the parent of σ in the interpretation, tag σ with b_k, and if $\bar{a}_k$ is a leaf in $\mathcal{T}_{\mathcal{F}}^{\mathcal{A}} \subseteq A^{<\omega}$, label it as a leaf here too with the same label. Let us remark that these relations in $\mathcal{T}_\alpha(\mathcal{A})$ can be defined using only the labeled-tree structure of $\mathcal{T}_\alpha(\mathcal{A})$ since all $\Sigma_1^{\mathfrak{c}}$ formulas about a tuple in $\mathcal{A}$ can be read off $\mathcal{T}_\alpha(\mathcal{A})$.

There is no need to consider the congruence relation as each branch is being repeated infinitely often. □

Theorem 3.4. *Suppose that a replicated labeled tree $\mathcal{S}$ is functorially reducible to a structure $\mathcal{A}$. Then $\mathcal{S}$ is functorially reducible to $\mathcal{T}_\infty(\mathcal{A})$.*

Proof. Consider an effective interpretation of $\mathcal{S}$ in $\mathcal{A}$ with domain $\mathcal{S}^{\mathcal{A}}$. We define an effective interpretation of $\mathcal{S}$ in $\mathcal{T}_\infty(\mathcal{A})$ whose domain is a subset of $\mathcal{T}_\infty(\mathcal{A})$. Put $\langle(\bar{a}_0, n_0), (\bar{a}_1, n_1), ...(\bar{a}_k, n_k)\rangle$ in the domain of the interpretation if each of $(n_i, \bar{a}_i)$ is in the domain of the interpretation with parent $(n_i, \bar{a}_{i-1})$, and label it with the same label it has in the interpretation. Again, we must remark that these relations in $\mathcal{T}_\infty(\mathcal{A})$ can be defined using only the labeled-tree structure of $\mathcal{T}_\infty(\mathcal{A})$ since all $\Sigma_1^{\mathfrak{c}}$ formulas about a tuple in $\mathcal{A}$ can be read off $\mathcal{T}_\infty(\mathcal{A})$. □

4 Two tools

In this section we describe two tools that we will use in our main construction.

4.1 C.e. labels

It is often useful in computable structure theory to use c.e. labels. See for instance [DKL+15]. These labels are part of the language as for any regular structure. What changes is that for a presentation of the structure to be called *computable* we only require these labels to be c.e. These structures can be interpreted by standard structures by adding a repository of elements that we call labels identified by a unary relations that identify which type of labels they are, and adding relations that attach a tuple to a label-element when we want to label the tuple. All label-elements must be used, and they can be used for a unique tuple.

To use c.e. labels on the tree of tuples we need to modify the way we label the tuples a little bit. An alternative definition for the tree of tuples would be the following: Consider the tree of tuples with the parent relation the same way as before. But instead of labeling each node with the finite atomic diagram of the tuple we do the following: We keep the same vocabulary we had for the structure; if we had an n-ary relation symbol R, we let it hold of a tuple $\langle\sigma_1, ..., \sigma_n\rangle$ in the tree if the σ_i all belong to the same path (i.e., no two of them are incomparable), and the relation holds of $\langle a_1, ..., a_n\rangle$, where a_i is the last element of the tuple σ_i. It is not hard to prove that this new tree of tuples is effectively bi-interpretable with the original one—they are essentially the same structure. The difference is that now, we can view some of these labels as c.e. labels. The tree of labels of a structure with c.e. labels is now a tree with c.e. labels.

A straightforward argument shows that if we first take the tree of tuples and then add label-elements we get a structure that is effectively bi-interpretable to what we would get if we first replace the c.e. labels by label-elements and then take the tree of tuples.

4.2 Joining trees

Later we will use the fact that if we have a disjoint union of structures, we can compute the back-and-forth tree of the disjoint union from the back-and-forth trees of each component.

Lemma 4.1. *Let $\mathcal{A}$ be the disjoint union of $\mathcal{A}_i$. Suppose that $\mathcal{T}(\mathcal{A}_i)$ has a computable copy uniformly in i. Then $\mathcal{T}(\mathcal{A})$ has a computable copy.*

Proof. To build $\mathcal{T}(\mathcal{A})$ out of $\{\mathcal{T}(\mathcal{A}_i) : i \in \omega\}$, consider the tree of tuples of elements from $\bigcup_i \mathcal{T}(\mathcal{A}_i)$, where the elements from the same tree must belong to the same path. □

5 Main Result

In this section, we will prove the main result of this paper.

Theorem 1.10. *There is a structure $\mathcal{A}$ with no computable copy such that $\mathcal{T}(\mathcal{A})$ has a computable copy.*

The proof is quite involved and will extend over the rest of this section. We build $\mathcal{A}$ as the disjoint union of structures $\mathcal{A}_n$, with $\mathcal{A}_n$ satisfying a unary relation R_n. We will make sure that $\mathcal{A}_n$ is not isomorphic to the structure with domain R_n within the nth computable structure. Thus $\mathcal{A}$ will have no computable copy. $\mathcal{T}(\mathcal{A}_n)$ will have a computable copy which is computable uniformly in n, and so by Lemma 4.1, $\mathcal{T}(\mathcal{A})$ will have a computable copy.

Fix n for which we will define $\mathcal{A} = \mathcal{A}_n$ which is not isomorphic to $\mathcal{B}$, the structure with domain R_n in the nth (possibly partial) computable structure. The language of $\mathcal{A}$ will consist of infinitely many unary relations called labels and infinitely many n-ary relations for each n. The relations R will be relations on unordered tuples. We will consider these relations to all be c.e. relative to a presentation of $\mathcal{A}$. Similarly, we will consider the trees to have c.e. labels.

We will build $\mathcal{A}$ as a Δ^0_2-limit of a computable sequence of finite structures on increasing domains. The construction will proceed by stages, each divided into steps. At stage s, step t, we will define $\mathcal{A}_{s,t}$. At each stage s and step t, after defining $\mathcal{A}_{s,t}$, we wait for an i such that $\mathcal{B}_i$ is isomorphic to $\mathcal{A}_{s,t}$. If there is no such stage, then we just put $\mathcal{A} = \mathcal{A}_{s,t}$ and win; in this case, there are only finitely many stages of the construction. If the construction lasts for infinitely many stages, then there will be a sequence $s_0 \prec s_1 \prec s_2 \prec \cdots$ of true stages, and we will have $\mathcal{A} = \bigcup_{s_i} \mathcal{A}_{s_i,0}$ as a nested union. When describing the construction we will generally assume that for each stage s and step t a value i as above exists, and write $\mathcal{B}_{s,t}$ for $\mathcal{B}_i$; note that the $\mathcal{B}_{s,t}$ are nested, with $\mathcal{B}$ being their union. It is this asymmetry—that the opponent must produce nested structures while we do not—that we will exploit to make $\mathcal{A}$ not isomorphic to $\mathcal{B}$; however, we will still have to show that a copy of $\mathcal{T}(\mathcal{A})$ is uniformly computable, whether there are finitely or infinitely many stages of the construction.

To each stage s we associate a number $n(s)$. The true stages are those stages s for which, for every $t \geq s$, $n(t) \geq n(s)$. If $s \leq t$, we say that t *believes* s *is a true stage*, or for short t believes s, and write $s \preceq t$, if for all $s < s' < t$, $n(s') \geq n(s)$. If t is a true stage and $s \leq t$, then t believes s if and only if s is a true stage. Along the construction, it will always be the case that either $n(s+1) = n(s)+1$ or $n(s+1) < n(s)$; the value of n can never stay the same from one stage to the next, or increase by more than one.

To show that $\mathcal{T}(\mathcal{A})$ is computable, we will build a computable sequence of labeled trees $T_{s,t}$, each isomorphic to the corresponding $\mathcal{T}(\mathcal{A}_{s,t})$, and such that

$$T_{0,0} \subseteq T_{0,1} \subseteq T_{0,2} \subseteq \cdots \subseteq T_{1,0} \subseteq T_{1,1} \subseteq \cdots \subseteq T_{2,0} \subseteq \cdots .$$

Since the $\mathcal{A}_{s,t}$ are not nested, it is not true that

$$\mathcal{T}(\mathcal{A}_{0,0}) \subseteq \mathcal{T}(\mathcal{A}_{0,1}) \subseteq \mathcal{T}(\mathcal{A}_{0,2}) \subseteq \cdots \subseteq \mathcal{T}(\mathcal{A}_{1,0}) \subseteq \mathcal{T}(\mathcal{A}_{1,1}) \subseteq \cdots \subseteq \mathcal{T}(\mathcal{A}_{2,0}) \subseteq \cdots$$

under the natural inclusion of their domains; so the structure $\mathcal{A}$ will have to be constructed carefully to ensure that the $T_{s,t}$ can be nested. The tree T which is the union of the $T_{s,t}$ would then be computable, but we must argue that it is actually isomorphic to $\mathcal{T}(\mathcal{A})$. To see this, we will also keep track of maps $g_{s,t} \colon T_{s,t} \to \mathcal{A}_{s,t}$. These maps will respect the relationship between the labels on the tree and the atomic types in the structure; we call such a map a *pseudoisomorphism*. To ensure that $g = \lim g_{s,0}$ is a pseudoisomorphism $T \to \mathcal{A}$, during the construction we will make sure that if $s \preceq s'$ then $g_{s',0}$ does not change too much from $g_{s,0}$; more precisely:

- if $s \preceq s'$ and $n(s) = n(s')$ then $g_{s,0} \subseteq g_{s',0}$; and
- if $s \preceq s'$ and $n = n(s) < n(s')$ then $g_{s,0}$ and $g_{s',0}$ agree on all nodes of $T_{s,0}$ of height $\leq n$.

Then there are three cases to consider:

1. There are only finitely many stages. If s and t are the last stage and step, then $\mathcal{A} = \mathcal{A}_{s,t}$ and $T = T_{s,t} \cong \mathcal{T}(\mathcal{A}_{s,t})$ so we are done.
2. There is n such that for some s_0, for all true stages $s \geq s_0$, $n(s) = n$. In this case, $g = \bigcup_{\text{true } s \geq s_0} g_{s,0}$ is a nested union.
3. There is no bound on $n(s)$ for s a true stage. Then the $g_{s,0}$ come to a limit g defined as follows. For $\sigma \in T$, $g(\sigma) = g_{s,0}(\sigma)$ where s is any true stage which is sufficiently large that $\sigma \in T_{s,0}$ and $n(s) > |\sigma|$. This g is a pseudoisomorphism $T \to \mathcal{A}$.

See Lemma 5.8 for the details.

At each stage s and step t, every element of $\mathcal{A}_{s,t}$ will have a label ℓ that holds of that element and no other. We call this ℓ the (s,t)-distinguishing label of a (or the s-distinguishing label if $t = 0$); as the notation implies, the distinguishing label of a particular element of $\mathcal{A}$ will change over time. At each stage s and step t, we will also have certain elements which are designated as *killed*. What this means is that these elements will never again be given

a new label, and they will keep the same distinguishing label. Once an element is killed it will remain killed from then on, even if it became killed at a non-true stage.

At stage s, let $n = n(s)$ and let $s_0 \prec s_1 \prec s_2 \prec \cdots \prec s_n = s$ be the previous stages believed by s such that s_i is the least stage with $n(s_i) = i$. The elements of $\mathcal{A}_{s,0}$ will consist of:

- c,
- $a_1, \ldots, a_n$,
- for each $1 \leq i < n$, $a'_{1,s_i}, \ldots, a'_{i,s_i}$ for each i, and
- killed elements.

These elements are all distinct. Over the course of stage s, we will introduce elements c'_s and $a'_{1,s}, \ldots, a'_{n,s}$. If $n(s+1) = n+1$ we will have $a_{n+1} = c'_s$, while if $n(s+1) < n$ we will kill a bunch of elements.

These names that we give the elements of $\mathcal{A}_{s,0}$ that are dependent on the stage. If $s \preceq t$, then t agrees with all of the values defined at stage s; thus all true stages agree on the values of these elements. But a stage s with $n = n(s)$ which is not a true stage may define a value for a_n which is later killed once we find out that s is not true, and then a later stage t with $n = n(t)$ may redefine a_n as a new element.

Suppose that the construction has infinitely many stages; we must argue that $\mathcal{A}$ is not isomorphic to $\mathcal{B}$. The key is the element $c \in \mathcal{A}$: we will ensure that $\mathcal{B}$ does not have an element isomorphic to it. Because $\mathcal{B}_{s,t}$ is isomorphic to $\mathcal{A}_{s,t}$, and each element of $\mathcal{A}_{s,t}$ has a distinguishing label, there is a unique isomorphism $f_{s,t}$ between $\mathcal{A}_{s,t}$ and $\mathcal{B}_{s,t}$. We may assume that the domain of $\mathcal{B}_{s,t}$ is an initial segment of ω. During the construction we will make sure that at every stage s, $f_{s,0}(a_1), \ldots, f_{s,0}(a_n) < f_{s,0}(c)$ in the standard order on ω; it is the opponent that decides how to build $\mathcal{B}$, so we will have to construct $\mathcal{A}$ in such a way that the opponent is forced to maintain this. Thus if, along the true stages, $\lim_s n(s) \to \infty$, then $\lim_s f_{s,0}(c) \to \infty$, which will allow us to ensure that $\mathcal{B}$ will have no element isomorphic to $c \in \mathcal{A}$. (To make this true, we also guarantee that if s is a true stage, c will never be given the s-distinguishing label of any other element of $\mathcal{A}_{s,0}$.) The other possibility is that for sufficiently large true stages s, the value of $n(s)$ stays the same, say $n(s) = n$. Whenever we have $s \preceq s'$ and $n(s) = n(s')$, we will have $f_{s,0}(c) < f_{s',0}(c)$; this will be because a new element shows up in $\mathcal{B}$ below the image of c, and then that new element is then killed. So in this case as well, we will have that $\mathcal{B}$ does not contain an element isomorphic to c.

5.1 First few stages of the construction

In an informal way we will go through the first few steps of the construction. In Figures 1 and 2 we show the first two stages of the construction, ending in stage 2 step 0. The two figures show the two possibilities, depending on how $\mathcal{B}$ responds. We always have $n(0) = 0$ and $n(1) = 1$ since we cannot have $n(1) < n(0) = 0$. But then we could either have $n(2) = 2$ in which case 0, 1, and 2 are all true stages (at least so far), or $n(2) = 0$, in which case 0

and 2 are true stages so far, but 1 is not. The figures show $\mathcal{A}_{s,t}$ for each stage, the response $\mathcal{B}_{s,t}$ by our opponent, and the tree $T_{s,t}$. The nodes of the tree are labeled with the preimages of the pseudoisomorphism $g_{s,t}\colon T_{s,t} \to \mathcal{A}_{s,t}$. Elements of the structures and the tree are represented by black dots; two dots which are in the same position from one diagram to the next represent the same elements of the domain. So, for example, the node on the first level of the tree at $0,1$ which is labeled c is the same as the node at $0,2$ labeled c'; what has happened is that the image of this node under the pseudoisomorphism has changed. In the structures $\mathcal{A}$ and $\mathcal{B}$, the numbers below an element represent the labels given to that element. The elements of $\mathcal{B}$ are labeled with their preimages under $f_{s,t}$ though we drop the subscript to save space.

Begin at stage 0 (and step 0) with $\mathcal{A}_0$ consisting of a single element c with the label '1'. At step 1, we introduce a new relation R_0 and have it hold of c; this is represented in the diagrams by the line attached to the element. At step 2, we introduce a new element c'_0 and change the structure to have R_0 hold of c'_0 instead of c. Both c and c'_0 have the same label '1' that c had at step 1, but they each get a new label that the other does not have: '2' for c and '3' for c'_0. Note that $\mathcal{A}_{0,1} \not\subseteq \mathcal{A}_{0,2}$. Now $\mathcal{B}$ must copy $\mathcal{A}$, except that as it has already put R_0 on the first element of its domain, this element must copy c'_0 rather than c. We must also expand the tree, and adjust the pseudoisomorphism; the node that previously mapped to c now maps to c'_0. Finally, at stage 1 step 0, we put R_0 on all of the elements of $\mathcal{A}$ and promise to put it on any new elements that show up from now on. This essentially means that we can forget about R_0 from now on, and we no longer draw it. We also set $n(1) = 1$ and $a_1 = c'_0$. Note that $f(a_1) < f(c)$, so that we have made one step towards forcing $\mathcal{B}$ to omit an image of the element $c \in \mathcal{A}$.

The objective of stage 1 is to get one of the following two outcomes: One is to create an element a_2 and end stage 1 with $f(a_1), f(a_2) < f(c)$ and $n(2) = 2$; The other possibility is to end the stage with $n(2) = 0$ in which case we need $T_{2,0}$ to extend $T_{0,0}$ preserving the pseudo-isomorphism, meaning that we would need to move c back—recall we had switch it with c' (later renamed a_1) in step $0,2$. For this we would need to give c and a_1 the same labels, risking that $\mathcal{B}$ can now switch the images of c and a_1. So, the only instance in which we would do this is if we have $f(c) < f(a_1)$ so that we do not mind if $\mathcal{B}$ switches them.

At stage 1 step 1, we put a new binary relation R_1 on c and a_1; this relation will be on unordered tuples. $\mathcal{B}_{1,1}$ must copy this. At step 2, we introduce new elements c'_1 and $a'_{1,1}$ which have the same labels that c and a_1 had respectively in $\mathcal{A}_{1,1}$. Each of the four elements gets a new label so that they are still distinguished. In $\mathcal{A}_{1,2}$, we put R_1 on the pairs $c, a'_{1,1}$ and c'_1, a_1 but not on c, a_1; so $\mathcal{A}_{1,1} \not\subseteq \mathcal{A}_{1,2}$. Before discussing our opponent's possible responses, let us talk about the tree $T_{1,2}$. Once again the pseudoisomorphism $T_{1,1} \to \mathcal{A}_{1,1}$ cannot be extended to a pseudoisomorphism $T_{1,2} \to \mathcal{A}_{1,2}$. However, we can keep the pseudoisomorphism the same on the first level of the tree, because the tuple c, a_1 satisfies the same atomic formulas in $\mathcal{A}_{1,1}$ as the tuples c'_1, a_1 and $c, a'_{1,1}$; what is going on here is that any existential formula true in $\mathcal{A}_{1,1}$ with one$(= n(1))$ parameter is still true of the same parameter in $\mathcal{A}_{1,2}$. Looking ahead: If at a later stage t we have $1 \prec t$ and $n(1) = 1 = n(t)$, we will need $g_{t,0}$ be a full extension of the pseudoisomorphism $T_{1,0}$ and not just of the first level. For that we will need to be able to switch the images of c and c'_1 (soon to be renamed a_2).

Our opponent must copy $\mathcal{A}_{1,2}$, and they have two choices; they can either turn what used to be the image of c into the image of c'_1, or they can turn what used to be the image of a_1

s,i	$\mathcal{A}_{s,i}$	$\mathcal{B}_{s,i}$	$T_{s,i}$
0,0	c ● 1	$f(c)$ ● 1	c ●
0,1	c ● 1	$f(c)$ ● 1	c ●
0,2	c ● c' ● $1,2$ $1,3$	$f(c')$ ● $f(c)$ ● $1,3$ $1,2$	c ● c' ● c' ● c ●
1,0	c ● a_1 ● $1,2$ $1,3$	$f(a_1)$ ● $f(c)$ ● $1,3$ $1,2$	c ● a_1 ● a_1 ● c ●
1,1	c ● a_1 ● $1,2$ $1,3$	$f(a_1)$ ● $f(c)$ ● $1,3$ $1,2$	c ● a_1 ● a_1 ● c ●
1,2	c ● a_1 ● c' ● $a'_{1,1}$ ● $1,2,4$ $1,3,5$ $1,2,6$ $1,3,7$	$f(a_1)$ ● $f(c')$ ● $f(c)$ ● $f(a'_{1,1})$ ● $1,3,5$ $1,2,6$ $1,2,4$ $1,3,7$	c' ● c ● $a'_{1,1}$ ● a_1 ● $a'_{1,1}$ ● a_1 ● c ● c' ●
2,0	c ● a_1 ● a_2 ● $a'_{1,1}$ ● $1,2,4$ $1,3,5$ $1,2,6$ $1,3,7$	$f(a_1)$ ● $f(a_2)$ ● $f(c)$ ● $f(a'_{1,1})$ ● $1,3,5$ $1,2,6$ $1,2,4$ $1,3,7$	a_2 ● c ● $a'_{1,1}$ ● a_1 ● $a'_{1,1}$ ● a_1 ● c ● a_2 ●

Figure 1: The case $n(2) = w$.

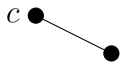
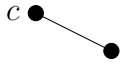
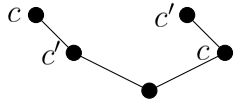
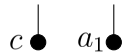
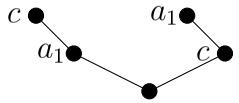
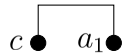
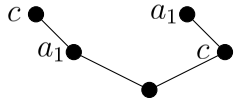
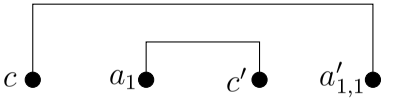
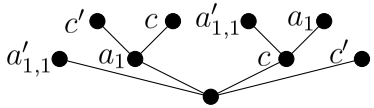
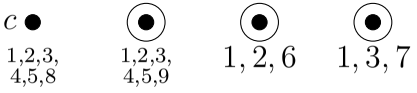
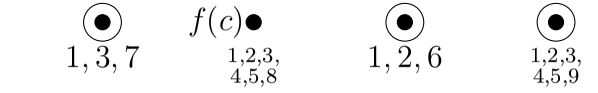
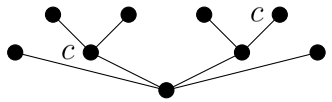
s,i	$\mathcal{A}_{s,i}$	$\mathcal{B}_{s,i}$	$T_{s,i}$
0,0			
0,1			
0,2			
1,0			
1,1			
1,2			
2,0			

Figure 2: The case $n(2) = 0$.

into the image of $a'_{1,1}$. In Figure 1 we show the former, and in Figure 2 we show the latter. We will begin by discussing the former. Since $f(c'_1) < f(c)$, at stage 2 step 0 we can set $a_2 = c'_1$, $n(2) = 2$, and put the relation R_1 on every pair of elements of $\mathcal{A}$ (so that we stop drawing it). Note that we have $f_{2,0}(a_1), f_{2,0}(a_2) < f_{2,0}(c)$ as desired. We have also preserved, from stage 1 to stage 2, the pseudoisomorphism on the first level of the tree.

Let us now consider the other possibility for the opponent's response $\mathcal{B}_{1,1}$, as shown in Figure 2. The key is that the opponent has put the image of a_1 after the image of c. In $\mathcal{A}_{2,0}$, we will do the following. First, we put R_1 on every pair of elements, and stop drawing it. Second, we take all of the labels on $c'_0 = a_1$ in $\mathcal{A}_{1,2}$ and give them to c, and vice versa. Both of these elements receive a new distinguishing label. Finally, we set $n(2) = 0$ and kill all of the elements that were introduced since stage 0, i.e., all of the elements other than c. In the diagrams, we use a circle to represent an element that has been killed.

Note that in the tree $T_{2,0}$, we are able to switch back the swap of c and c'_0 that we made at stage 0 step 2. This is because each of these elements satisfies in $\mathcal{A}_{0,2}$ all of the same existential formulas the other satisfied in $\mathcal{A}_{1,2}$. Thus the pseudoisomorphism $T_{2,0} \to \mathcal{A}_{2,0}$ extends the pseudoisomorphism $T_{0,0} \to \mathcal{A}_{0,0}$ at the previous 2-true stage. The unlabeled elements of the tree are mapped to a killed element, and so will never have to change from now on.

The opponent is also allowed to switch, from $\mathcal{B}_{1,2}$ to $\mathcal{B}_{2,0}$, the images of c and $c'_0 = a_1$. However, because the opponent made the image of $c'_0 = a_1$ larger than the image of c, this will not allow them to decrease the image of c. So essentially we have returned to where we were at stage 0, except that we have some new elements which are all killed (and hence will not interfere with the construction) and that the image of c in $\mathcal{B}_{2,0}$ has increased compared to what it was in $\mathcal{B}_{0,0}$, so that we have made some progress towards having $\mathcal{A} \ncong \mathcal{B}$.

An additional consideration. At stage s where $n(s) > 2$, it is possible that we will require more steps. We illustrate this in Figure 3. Due to space constraints, we will no longer keep track of the trees $T_{s,t}$. We begin at stage 2 step 0 where we left off in the case $n(2) = 2$ as in Figure 1. We begin by introducing a ternary relation R_2 in step 1, followed by elements c'_2, $a'_{1,2}$, and $a'_{2,2}$ in step 2. Suppose that the opponent responds with $\mathcal{B}_{2,2}$ as shown. To take advantage of the fact that they have put the image of a_2 above that of c—and so the image of the new element $a'_{2,2}$ is below c—in $\mathcal{A}_{2,3}$ we give c every label that a_2 had in $\mathcal{A}_{2,2}$, and vice versa. We also give a_1 every label that $a'_{1,1}$ had in $\mathcal{A}_{2,2}$. This allows us to undo the injury to the pseudoisomorphism $T_{s,t} \to \mathcal{A}_{s,t}$ that was done at stage 1 step 2 (on the second level of the tree, see Figure 1). Now the opponent, in building $\mathcal{B}_{2,3}$, can swap the images of c and $a_2 = c'_1$, and of a_1 and $a'_{1,1}$. One way that they might do so is shown in Figure 3. The issue here is that we have allowed our opponent to make $f(c) < f(a_1)$, and so we cannot go on to the next stage. But what we can do is a similar move again: give c all of the labels that a_1 had in $\mathcal{A}_{2,3}$ and vice versa. This allows us to undo the injury done to the pseudo-isomorphism in stage 0 step 2, and we have $n(3) = 0$.

5.2 Formal Construction of $\mathcal{A}$

The formal construction will look slightly different than the informal picture just given. At a stage s, we may not know how many steps there will be until $\mathcal{B}$ allows us to move on to

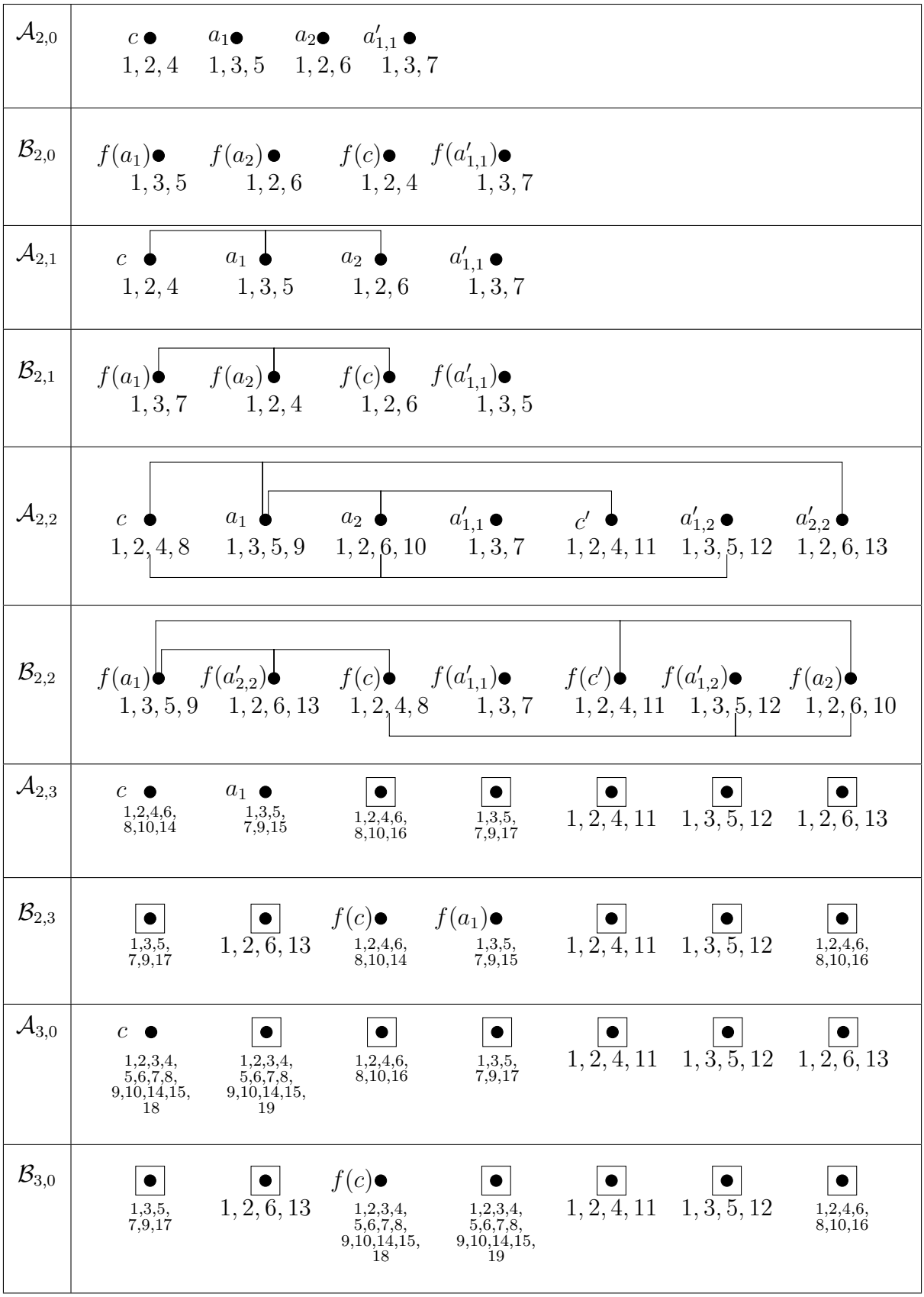


Figure 3: One possibility for stage 2.

the next stage. Because of this, if t is the last step at stage s, we will have $\mathcal{A}_{s+1,0} = \mathcal{A}_{s,t}$.

We will also keep track, through the steps t at stage s, of a value $m_{s,t}$. This value will be the current guess at $n(s+1)$. If t is the last stage of step s, then we will have $n(s+1) = m_{s,t}$.

Construction.

- *Stage* $s = 0$*, step* 0. Begin with $\mathcal{A}_0$ consisting of a single element c labeled with a single label.
- *Stage* $s+1$*, step* 0. Let t be the last step at stage s. Let $\mathcal{A}_{s+1,0} = \mathcal{A}_{s,t}$. Let $n(s+1) = m_{s,t}$. If $n(s+1) > n(s)$, let $a_{n(s+1)} = c'_s$. Otherwise, kill (meaning leave as is and ignore from this point on) everything but the values:
  - c,
  - $a_1, \ldots, a_{n(s+1)}$,
  - $a'_{1,s_i}, \ldots, a'_{i,s'_i}$ for each $1 \leq i < n(s+1)$ where s_i is the last stage with $s_i \prec s+1$ and $n(s_i) = i$.
- *Stage* s*, step* 1. Let $n = n(s)$. Choose a new $n+1$-ary relation R_s. $\mathcal{A}_{s,1}$ will be $\mathcal{A}_{s,0}$ except that we have
$$R_s(c, a_1, \ldots, a_n).$$
Recall that R_s is a relation on unordered tuples.
- *Stage* s*, step* 2. Since we did not add any new elements or labels in $\mathcal{A}_{s,1}$, we have that $f_{s,1} = f_{s,0}$. So we have $\mathcal{B}_{s,1} \models R_s(f_{s,0}(c), f_{s,0}(a_1), \ldots, f_{s,0}(a_n))$.

  The structure $\mathcal{A}_{s,2}$ will extend $\mathcal{A}_{s,0}$ by adding new elements, labels, and relations. Introduce, in $\mathcal{A}_{s,2}$, new elements $c'_s, a'_{1,s}, \ldots, a'_{n,s}$. These elements have the same labels that the corresponding elements $c, a_1, \ldots, a_n$ had in $\mathcal{A}_{s,0}$, but we give each element a new unique label. So, for example, c and c'_s have all the same labels that c had in $\mathcal{A}_{s,0}$, but each of them has one more label that the other does not. We have relations
$$R_s(c'_s, a_1, \ldots, a_n) \qquad \text{and, for each } i, \qquad R_s(c, a_1, \ldots, a'_{i,s}, \ldots, a_n).$$
We do not have $R_s(c, a_1, \ldots, a_n)$ anymore. In general, R_s holds of any such tuple where there is exactly one $'$.
- *Stage* s*, step* 3. Whether we proceed onto stage $s+1$, or add another step to stage s, depends on how $\mathcal{B}$ responds.

  Recall that $f_{s,1} = f_{s,0}$ was the isomorphism $\mathcal{A}_{s,1} \to \mathcal{B}_{s,1}$, and so it gives a map $\mathcal{A}_{s,2} \to \mathcal{B}_{s,2}$, but this map will not be an isomorphism since $\mathcal{A}_{s,1}$ is not a substructure of $\mathcal{A}_{s,2}$ while $\mathcal{B}_{s,1}$ is a substructure of $\mathcal{B}_{s,2}$. There are two possibilities for $f_{s,2}(c)$; either it is equal to $f_{s,0}(c)$, or it is one of the new elements in $\mathcal{B}_{s,2} \setminus \mathcal{B}_{s,1}$. This is because all of the other elements of $\mathcal{B}_{s,1}$ have a label which c does not. Similarly, for each a_i, either $f_{s,2}(a_i)$ is $f_{s,0}(a_i)$ or it is one of the new elements in $\mathcal{B}_{s,2} \setminus \mathcal{B}_{s,1}$.

Since $\mathcal{B}_{s,1}$ is a substructure of $\mathcal{B}_{s,2}$, we must have that

$$\mathcal{B}_{s,2} \models R_s(f_{s,0}(c), f_{s,0}(a_1), \ldots, f_{s,0}(a_n))$$

since this was true of $\mathcal{B}_{s,1}$. However, $\mathcal{A}_{s,2} \not\models R_s(c, a_1, \ldots, a_n)$. So the isomorphism $f_{s,2}\colon \mathcal{A}_{s,2} \to \mathcal{B}_{s,2}$ cannot extend $f_{s,0}$. There are $n+1$ different $n+1$-tuples which satisfy R_s in $\mathcal{A}_{s,2}$, and it must be one of these that maps to $f_{s,0}(c), f_{s,0}(a_1), \ldots, f_{s,0}(a_n)$ in $\mathcal{B}_{s,2}$. These tuples are $c'_s, a_1, \ldots, a_n$ and, for each i, $c, a_1, \ldots, a'_{i,s}, \ldots, a_n$; moreover, these must map to $f_{s,0}(c), f_{s,0}(a_1), \ldots, f_{s,0}(a_n)$ in order. So we have $n+1$ possibilities divided into two cases:

Case 1: $f_{s,2}(c) \in \mathcal{B}_{s,2} \setminus \mathcal{B}_{s,1}$. For each $1 \leq i \leq n$, $f_{s,2}(a_i) = f_{s,0}(a_i) \in \mathcal{B}_{s,1}$.

Case 2: There is j such that $f_{s,2}(a_j) \in \mathcal{B}_{s,2} \setminus \mathcal{B}_{s,1}$. For each $1 \leq i \leq n$, $i \neq j$, $f_{s,2}(a_i) = f_{s,0}(a_i) \in \mathcal{B}_{s,1}$; and $f_{s,2}(c) = f_{s,0}(c) \in \mathcal{B}_{s,1}$.

In Case 1: This is the last step of stage s. We define $\mathcal{A}_{s,3} \supseteq \mathcal{A}_{s,2}$. Put R_s on every $n+1$-tuple of elements, and on every $n+1$-tuple of elements which is added to $\mathcal{A}$ at any later stage of the construction. Set $m_{s,3} = n(s) + 1$.

Note that, since $f_{s,2}(c'_s) = f_{s,0}(c)$, we have $f_{s,2}(a_1) < f_{s,2}(a_2) < \cdots < f_{s,2}(a_n) < f_{s,2}(c'_s) < f_{s,2}(c)$, and hence we are in position to define $a_{n+1} = c'_s$ in the next stage.

In Case 2: We do not yet move onto stage $s+1$. We define $\mathcal{A}_{s,3} \supseteq \mathcal{A}_{s,2}$. Put R_s on every $n+1$-tuple of elements, and on every $n+1$-tuple of elements which is added to $\mathcal{A}$ at any later stage of the construction.

Let $s' < s$ be the previous stage with $n(s') = j - 1$. Give c all of the labels $a_j = c'_{s'}$ had in $\mathcal{A}_{s,2}$, and give $a_j = c'_{s'}$ all of the labels c had in $\mathcal{A}_{s,2}$. Similarly, for $1 \leq i < j$, give a_i all of the labels $a'_{i,s'}$ had in $\mathcal{A}_{s,2}$, and give $a'_{i,s'}$ all of the labels a_i had in $\mathcal{A}_{s,2}$. Each of these elements gets a new $(s,3)$-distinguishing label.

What this does is allow us to define the pseudoisomorphism $T_{s,3} \to A_{s,3}$ extending the pseudoisomorphism we had at stage s' by allowing us to swap back the change we made in step 2 of stage s'. The risk is that $\mathcal{B}$ might now swap c and $c'_{s'}$ (which was renamed to a_j), but that is not a worry since $f_{s,2}(c) < f_{s,2}(a_j)$.

Set $m_{s,3} = j - 1$. Continue the stage s with step 4 of stage s.

- *Stage s, step $t+1 > 3$.* Let $m = m_{s,t}$. We have two cases.

  Case $(t.1)$**:** $f_{s,t}(a_1), \ldots, f_{s,t}(a_m) < f_{s,t}(c)$.

  Case $(t.2)$**:** For some j, $f_{s,t}(a_j) > f_{s,t}(c)$.

  In Case $(t.1)$ *:* Step $t+1$ will be the last step of stage s. We define $\mathcal{A}_{s,t+1} = \mathcal{A}_{s,t}$. Set $m_{s,t+1} = m_{s,t}$.

  In Case $(t.2)$*:* Let j be least such that $f_{s,t}(a_j) > f_{s,t}(c)$. Let $s' < s$ be the previous stage with $n(s') = j - 1$. Give c all of the labels $a_j = c'_{s'}$ had in $\mathcal{A}_{s,t}$, and give $c'_{s'}$ all of the labels c had in $\mathcal{A}_{s,t}$. Similarly, for $i < j$, give a_i all of the labels $a'_{i,s'}$ had in $\mathcal{A}_{s,t}$, and give $a'_{i,s'}$ all of the labels a_i had in $\mathcal{A}_{s,t}$. Each of these elements gets a new $(s, t+1)$-distinguishing label.

  Set $m_{t+1} = j - 1$. Continue the construction with step $t+2$ of stage s.

End of construction

5.3 Verification

We will begin by proving various lemmas about the construction. First, we show that elements that are introduced at non-true stages are killed when we find out that the stage was not true.

Lemma 5.2. *Suppose that $s \not\preceq s+1$, and let $s' \leq s$ be the previous $s+1$-true stage. Then each element of $\mathcal{A}_s \setminus \mathcal{A}_{s'}$ is killed by stage $s+1$.*

Proof. At stage $s+1$ step 0, we kill all of the elements other than:

- c,
- $a_1, \ldots, a_{n(s+1)}$,
- $a'_{1,s_i}, \ldots, a'_{i,s'_i}$ for each $1 \leq i < n(s+1)$ where s_i is the last stage with $s_i \prec s+1$ and $n(s_i) = i$.

These elements were all introduced at stage s' or earlier. □

Lemma 5.3. *Suppose that $s \leq t$ and $x \in \mathcal{A}$ is killed at stage s. Then $f_s(x) = f_t(x)$.*

Proof. Looking at the construction, we see that no new labels are added to a killed element, and so the s-distinguishing label of x is still its t-distinguishing label. □

Lemma 5.4. *Suppose that $s \prec s+1 \prec t$. Then, for each $1 \leq i \leq n(s)$:*

$$f_t(a'_{i,s}) = f_{s+1}(a'_{i,s}) > f_s(c).$$

Proof. Since $s \prec s+1$, we must be in Case 1 at stage s step 3. For each i, $1 \leq i \leq n(s)$, there are only two elements of $\mathcal{A}_{s,2}$ satisfying the s-distinguishing label of a_i, namely a_i and $a'_{i,s}$, and of these only a_i was in $\mathcal{A}_{s,0}$. So there is only one element of $\mathcal{B}_{s,0}$ satisfying this label, and this element is $f_{s,0}(a_i)$. Since we are in Case 1, $f_{s,2}(a_i) = f_{s,0}(a_i)$ and so $f_{s,2}(a'_{i,s}) \in \mathcal{B}_{s,2} \setminus \mathcal{B}_{s,0}$; thus $f_{s,2}(a'_{i,s}) > f_{s,0}(c)$ since the latter is in $\mathcal{B}_{s,0}$. We also have $f_{s+1}(a'_{i,s}) = f_{s,2}(a'_{i,s})$ since we make no further changes to $\mathcal{A}$ at stage s. So we have shown that $f_{s+1}(a'_{i,s}) > f_s(c)$.

Now we claim that $a'_{i,s}$ has no more labels at stage t than it had at stage $s+1$. There are only two times when we give an element a new label at a stage u:

1. When we introduce the elements c'_u and $a'_{1,u}, \ldots, a'_{n(u),u}$, we give a new label to c and $a_1, \ldots, a_{n(u)}$.
2. In Cases 2 and (t.2) when we give c and $a_j = c'_{s'}$ each other's labels, and a_i and $a'_{i,s'}$ each other's labels, for some $s' < u$ with $n(s') = j-1$.

No new labels are given to $a'_{i,s}$ for the sake of (1). At each stage u, $s+1 \leq u < t$, since s is a u-true stage, (2) can only happen for $j > n(s)$. Thus a new label cannot be added to $a'_{i,s}$ for the sake of (2) before stage t. So $a'_{i,s}$ is still the only element at stage t satisfying its $s+1$-distinguishing label. This means that $f_t(a'_{i,s}) = f_{s+1}(a'_{i,s})$. □

Lemma 5.5. *For all $s \leq t$, $f_s(c) \leq f_s(t)$, and if $s \prec t$, $f_s(c) < f_s(t)$.*

Proof. We argue inductively. It suffices to show that if $s \prec s+1$, then $f_s(c) < f_{s+1}(c)$, and that if $t \not\prec s+1$, then $f_s(c) \leq f_{s+1}(c)$.

First, suppose that $s \prec s+1$. Then we must be in Case 1 at stage s step 3, as in Case 2 we end up with $n(s+1) < n(s)$. Then $f_{s+1}(c) \in \mathcal{B}_{s,2} \setminus \mathcal{B}_{s,1}$ and so $f_{s+1}(c) > f_s(c)$ as $f_s(c) \in \mathcal{B}_{s,1}$ and this is an initial segment of $\mathcal{B}_{s,2}$.

Second, suppose that $s \not\prec s+1$. We must be in Case 2 at stage s step 3. Note that we then have $f_{s,2}(c) = f_{s,0}(c) = f_s(c)$. We also have $f_{s,2}(c'_s) \in \mathcal{B}_{s,2} \setminus \mathcal{B}_{s,1}$ and so $f_{s,2}(c'_s) > f_{s,2}(c)$. Looking at what we did in the construction in Case 2, we see that either $f_{s,3}(c) = f_{s,2}(c)$ or $f_{s,3}(c) = f_{s,2}(c'_s) > f_{s,2}(c)$ as there is a label that at this point has been given only to c and c' in $\mathcal{A}$, and $f_{s,2}(c)$ and $f_{s,2}(c')$ in $\mathcal{B}$. In either case, $f_{s,3}(c) \geq f_{s,2}(c) = f_s(c)$.

The construction now proceeds to step 4. If, at step 4, we are in Case $(t.1)$, then we make no further changes to $\mathcal{A}$, and so $f_{s+1}(c) = f_{s,3}(c) \geq f_s(c)$ as desired. On the other hand, suppose that at step 4 we are in Case $(s.2)$. We will argue inductively on steps $r \geq 3$ that $f_{s,r}(c) \geq f_s(c)$, and then if r is the last step of stage s, $f_{s+1}(c) = f_{s,r}(c) \geq f_s(c)$. We already have the base case $r = 3$. Suppose that we know that $f_{s,r}(c) \geq f_s(c)$. At step $r+1$, if we are in Case $(t.1)$, then $r+1$ is the last step of stage s and we do nothing to $\mathcal{A}$, so that $f_{s+1}(c) = f_{s,r+1}(c) = f_{s,r}(c) \geq f_s(c)$. So suppose that we are in Case $(t.2)$. Let j be as in Case $(t.2)$, with $m_{r+1} = j-1$. We have $f_{s,r}(a_j) > f_{s,r}(c)$. Looking at the construction, we have either $f_{s,r+1}(c) = f_{s,r}(c)$ or $f_{s,r+1}(c) = f_{s,r}(a_j) > f_{s,r}(c)$, as these are the only elements of $\mathcal{A}_{s,r+1}$ with the (s,r)-distinguishing label of c. In either case, $f_{s,r+1}(c) \geq f_{s,r}(c)$ as desired. □

Lemma 5.6. *Suppose that s is a true stage, $x, y \in \mathcal{A}_{s,0}$ with $x \neq y$. Then no element of $\mathcal{A}$ has both the s-distinguishing label of x and the s-distinguishing label of y.*

Proof. The elements of $\mathcal{A}_{s,0}$ are:

- c,
- $a_1, \ldots, a_n$,
- for each $1 \leq i < n$, $a'_{1,s_i}, \ldots, a'_{i,s_i}$ for each i, and
- killed elements

where $n = n(s)$ and $s_0 \prec s_1 \prec s_2 \prec \cdots \prec s_n = s$ are the previous stages believed by s such that s_i is the least stage with $n(s_i) = i$.

Killed elements keep the same distinguishing labels forever, so we may assume that x and y are not killed. There are two times when we give a label already applied to one element to another element at a stage t:

1. When we introduce the elements c'_t and $a'_{1,t}, \ldots, a'_{n(t),t}$ which have the distinguishing labels of c and $a_1, \ldots, a_n$ respectively.
2. In Cases 2 and $(t.2)$ when we give c and $a_j = c'_{s'}$ each other's labels, and a_i and $a'_{i,s'}$ each other's labels, for some $s' < t$ with $n(s') = j-1$.

After stage s, since s is a true stage, (2) can only happen for $j > n(s)$ and $s' \geq s$. Thus the elements $a'_{1,s_i}, \ldots, a'_{i,s_i}$ for $1 \leq i < n$ keep the same distinguishing labels from stage s on.

We claim that the only elements which can receive the s-distinguishing label of a_i, $1 \leq i \leq n(s)$, are elements of the form $a'_{i,s'}$ for $s' \geq s$. Indeed we see that in both (1) and (2), whenever one of these elements (a_i or $a'_{i,s'}$ for $s' \geq s$) receives a label which had already been given to another element, that other element is also one of these elements. Thus no element can ever be given the s-distinguishing labels of both a_i and a_j, $i \neq j$, or of a_i and c. This proves the lemma. □

Lemma 5.7. *$\mathcal{A}$ is not isomorphic to $\mathcal{B}$.*

Proof. Let $s_0 \prec s_1 \prec s_2 \prec s_3 \prec \cdots$ be the true stages. Given $x \in \mathcal{B}$, we claim that x is not the isomorphic image of c. Using Lemma 5.5, let i be such that $x \in \mathcal{B}_{s_i,0}$ and $f_{s_i}(c) > x$. Then $x = f_{s_i}(y)$ for some $y \in \mathcal{A}_{s_i,0}$, $y \neq c$. So in $\mathcal{B}_{s_i,0}$, x has the s_i-distinguishing label of y. By Lemma 5.6, no element of $\mathcal{A}$ has both the s_i-distinguishing label of y and the s_i-distinguishing label of c. So x cannot be the isomorphic image of c. Thus $\mathcal{B}$ is not isomorphic to $\mathcal{A}$. □

Lemma 5.8. *$\mathcal{T}(\mathcal{A})$ has a computable copy.*

Proof. For each stage s, let i_s be the last step of stage s. We will computably build

$$T_{0,0} \subseteq T_{0,1} \subseteq \cdots \subseteq T_{0,i_0} = T_{1,0} \subseteq \cdots \subseteq T_{1,i_1} = T_{2,0} \subseteq \cdots$$

such that $T_{s,i}$ is isomorphic to $\mathcal{T}(\mathcal{A}_{s,i})$. Moreover, for each stage s, we have a pseudo-isomorphism $g_s \colon T_{s,0} \to \mathcal{A}_{s,0}$. We will argue that $T = \bigcup T_{s,i}$ is isomorphic to $\mathcal{T}(\mathcal{A})$.

Note that we define $T_{s,i}$ at every stage s and *step* i, whereas g_s is only defined for every *stage* s. It is of course possible that there are only finitely many stages, because $\mathcal{B}$ stops copying $\mathcal{A}$. Then, if s, i is the last stage and step, we have $T = T_{s,i}$ is isomorphic to $\mathcal{T}(\mathcal{A}_{s,i}) = \mathcal{T}(\mathcal{A})$. So T is isomorphic to $\mathcal{T}(\mathcal{A})$ even though we did not necessarily build an isomorphism between the two.

If there are infinitely many stages of the construction, then $T = \bigcup T_{s,0}$ will be an infinite union, so we must give an argument that T is isomorphic to $\mathcal{T}(\mathcal{A})$. We will have that $g = \lim g_s$ is a pseudo-isomorphism from $T \to \mathcal{A}$. This limit will be a Δ^0_2 limit along the true stages.

We make a new convention: At stage s, we call the elements $c, a_1, \ldots, a_{n(s)}$ the *active elements* of $\mathcal{A}_s$. The other elements are all inactive. We write $\overline{g}(\sigma)$ for the tuple $g(\sigma \restriction_1), \ldots, g(\sigma \restriction_{|\sigma|})$.

Given stages $s \preceq t$, the maps g_s and g_t will satisfy the following agreement condition:

(∗) If $n(s) = n(t)$ then $g_s \subseteq g_t$. If $n(t) > k = n(s)$ then whenever $\sigma \in T_{s,0}$ is such that $\overline{g}_s(\sigma)$ contains at most k active elements (but possibly other inactive or killed elements), then $g_s(\sigma) = g_t(\sigma)$.

Using (∗) we can argue that $f = \lim_s g_s$ is a pseudo-isomorphism $T \to \mathcal{A}$. Let $s_0 \prec s_1 \prec s_2 \prec \cdots$ be the sequence of true stages. If there are n and I such that for all $i \geq I$ $n = n(s_i)$, then $f = \bigcup_{i \geq I} g_{s_i}$ and these are nested. So we may assume that $\lim_{i \to \infty} n(s_i) = \infty$.

- Given $\sigma \in T$, we must argue that $g_s(\sigma)$ comes to a limit. Let s be a true stage with $\sigma \in T_{s,0}$ and $n(s) = k > |\sigma|$. Since $\bar{g}(\sigma)$ contains at most k elements, using $(*)$, we can argue inductively that $g_s(\sigma) = g_t(\sigma)$ for all true stages $t \geq s$. So $g(\sigma)$ comes to a limit on the true stages.

- Given $\bar{a} \in \mathcal{A}^{<\omega}$, we must argue that there is $\sigma \in T$ such that $g(\sigma) = \bar{a}$. Let s_i be a true stage such that $n(s_i) > |\bar{a}|$. Then $\bar{a}$ can have at most $|\bar{a}| < n(s_i)$ active elements. There is σ such that $g_{s_i}(\sigma) = \bar{a}$, and by $(*)$, $g(\sigma) = \bar{a}$.

We must now give the construction of T and f and then verify that they satisfy $(*)$. We will also build intermediate maps $g_{s,t} \colon T_{s,t} \to \mathcal{A}_{s,t}$, with $g_s = g_{s,0}$.

Construction. Begin with $T_{0,0} = \mathcal{T}(\mathcal{A}_{0,0})$ and g_0 the natural pseudo-isomorphism $T_{0,0} \to \mathcal{A}_{0,0}$. Suppose that we have defined

$$T_{0,0} \subseteq T_{0,1} \subseteq \cdots \subseteq T_{0,i_0} = T_{1,0} \subseteq \cdots \subseteq T_{1,i_1} = T_{2,0} \subseteq \cdots \subseteq T_{s,0}$$

and $g_0, \ldots, g_s$. We must define $T_{s,1}, T_{s,2}, \ldots, T_{s,i_s}, T_{s+1,0}$ and g_{s+1}. In the process, we will also define the $g_{s,t}$.

Step 1: $\mathcal{A}_{s,1} \supseteq \mathcal{A}_{s,0}$, and they have the same domains, so we can define $T_{s,1} \supseteq T_{s,0}$ with $g_{s,1} = g_{s,0} = g_s$ still a pseudo-isomorphism $T_{s,1} \to \mathcal{A}_{s,1}$.

Step 2: Recall that $\mathcal{A}_{s,2}$ did not extend $\mathcal{A}_{s,1}$: At step 1 we had $\mathcal{A}_{s,1} \models R_s(c, a_1, ..., a_n)$, while at step 2 we remove this relation and instead have $\mathcal{A}_{s,2} \models R_s(c', a_1, ..., a_n)$ and $\mathcal{A}_{s,2} \models R_s(c, a_1, ..., a'_i, ..., a_n)$. So we will not be able to define $g_{s,2}$ extending $g_{s,1}$, and we will have to settle for preserving only the strings with $n = n(s)$ active elements. The idea is that if a string $\sigma \in T_{s,1}$ maps to the tuple that contains all active elements $c, a_1, ..., a_n$, then we swap the active element in the tuple for its primed version. Here is the formal definition:

For each $\sigma \in T_{s,1}$, define $g_{s,2}(\sigma)$ as follows. If $g_s(\sigma)$ is inactive or killed, then $g_{s,2}(\sigma) = g_s(\sigma)$. If, among $g_s(\tau)$, for $\tau \prec \sigma$, there are less than n active elements, then also set $g_{s,2}(\sigma) = g_s(\sigma)$. Finally, if there are n active elements among $g_s(\tau)$, for $\tau \prec \sigma$, then set $g_{s,2}(\sigma)$ as follows: if $g_s(\sigma) = c$, then $g_{s,2}(\sigma) = c'_s$; if $g_s(\sigma) = a_i$, then $g_{s,2}(\sigma) = a'_{i,s}$.

We can see from the construction of $\mathcal{A}_{s,2}$ that this is a pseudoembedding. Now extend $T_{s,1}$ to $T_{s,2}$ and extend $g_{s,2}$ to a pseudoisomorphism $T_{s,2} \to \mathcal{A}_{s,2}$.

Step 3: If $s \preceq s+1$, then this is the last step of stage 3 and $\mathcal{A}_{s,3} = \mathcal{A}_{s,2}$. So set $g_{s,3} = g_{s,2}$.

If $s \not\preceq s+1$, then we have $n(s+1) < n(s)$ and we need g_{s+1} to extend the pseudoisomorphism build at the previous true stage. We do not know yet what the value of $n(s+1)$ is going to be, but our best current approximation is $m_{s,3}$ which is $< n(s)$. Let t be maximal with $n(t) = m_{s,3}$. Then we have $m_{s,3} = n(t) < n(t+1) \leq n(s)$ and $t \preceq s$. We want to build $g_{s,3}$ extending $g_t \colon T_{t,0} \to \mathcal{A}_{t,0}$.

We claim that $g_t \colon T_{t,0} \subseteq T_{s,2} \to \mathcal{A}_{s,3}$ is still a pseudoembedding. Consider $\sigma \in T_{t,0}$. $\bar{g}_t(\sigma)$ has at most $n(t)+1$ active members, as those are how many there are in $\mathcal{A}_{t,0}$. By the time we reach g_{t+1}, we might have swapped some elements but we did not add

any active element to $\bar{g}_{t+1}(\sigma)$. (We may swap c an c'_t which then is renamed as $a_{n(t)+1}$, but we could not have both in $\bar{g}_{t+1}(\sigma)$.) Therefore, $\bar{g}_{t+1}(\sigma)$ has at most $n(t+1)$ active members and hence, by $(*)$, it never changes again before stage $s+1$. So g_{t+1} is a pseudoembedding $T_{t,0} \to \mathcal{A}_{s,3}$.

We want to argue that also g_t is a pseudoembedding $T_{t,0} \to \mathcal{A}_{s,3}$. Given $\sigma \in T_{t,0}$, since $t \preceq t+1$ the only change we made from g_t to g_{t+1} is that sometimes when $g_t(\sigma) = c$ we set $g_{t+1}(\sigma) = c'_t = a_{n(t+1)}$, and sometimes when $g_t(\sigma) = a_i$ we set $g_t(\sigma) = a'_{i,t}$. But in Case 2 of step 3 at stage $s+1$, we give c all of the labels c'_t had and vice versa, and similarly for a_i and $a'_{i,t}$. So g_t is a pseudoembedding $T_{t,0} \to \mathcal{A}_{s,3}$.

Define $T_{s,3}$ and $g_{s,3} \colon T_{s,3} \to \mathcal{A}_{s,3}$ such that $g_{s,3} \supseteq g_t$ is a pseudoisomorphism.

Step t: This is similar to step 3.

Verification. We must now check that $(*)$ holds. For $(*)$, if $s \preceq s+1 = t$, then this is clear from the construction—step 2 above is the only place where g_{s+1} is made to differ from g_s. Otherwise, if $t < s$ is maximal such that $n(t) = n(s)$, then in step 3 / step t we define g_{s+1} by having it extend g_t. The other cases of $(*)$ follow from these two cases. □

5.4 Conclusion of the proof

We have built $\mathcal{A}_n$ (which, by abuse of notation, we were calling $\mathcal{A}$). By Lemma 5.7 is not isomorphic to $\mathcal{B}$, the structure with domain R_n in the nth (possible partial) computable structure. Moreover, by Lemma 5.8, $\mathcal{T}(\mathcal{A}_n)$ has a computable copy uniformly in n.

Then let $\mathcal{A}$ be the structure which is the disjoint union of the $\mathcal{A}_n$, each of which satisfies the unary relation R_n. Then $\mathcal{A}$ has no computable presentation. By Lemma 4.1, $\mathcal{T}(\mathcal{A})$ has a computable copy.

6 Jump inversions

We can use α-jump inversions to obtain the following corollary from Theorem 1.10.

Corollary 1.5. *For any computable ordinal α, there is a structure $\mathcal{A}$ such that $\mathcal{T}(\mathcal{A})$ has a computable copy but $\mathcal{A}$ itself has no Δ^0_α copy.*

We use the α-jump inversions as described in [GHK+05]. Let $\alpha > \omega$ be a computable successor ordinal. (We ask that $\alpha \geq \omega$ for simplicity just so that Δ^0_α and $0^{(\alpha)}$-computable coincide.) Fix computable structures $\mathcal{B}_0$ and $\mathcal{B}_1$ in the same relational language such that:

1. the pair $\{\mathcal{B}_0, \mathcal{B}_1\}$ is α-friendly,
2. $\mathcal{B}_0$ and $\mathcal{B}_1$ satisfy the same Σ_β sentences for $\beta < \alpha$,
3. each of $\mathcal{B}_0$ and $\mathcal{B}_1$ satisfies a computable Σ_α sentence that the other does not.

We may assume the vocabulary of $\mathcal{B}_0$ and $\mathcal{B}_1$ contains only one binary relation, and we may assume it is reflexive. What makes these structures useful is the following result: for every Δ^0_α set $S \subseteq \omega$, there is a computable sequence of computable structures $\{\mathcal{C}_i : i \in \omega\}$ such that $\mathcal{C}_i \cong \mathcal{B}_0$ if $i \notin S$ and $\mathcal{C}_i \cong \mathcal{B}_1$ if $i \in S$.

Given a structure $\mathcal{A}$, define the α-th jump inversion $\mathcal{A}^{(-\alpha)}$ of $\mathcal{A}$ as follows. Let $\{R_i : i \in I\}$ be the vocabulary of $\mathcal{A}$, with R_i of arity a_i. The domain of $\mathcal{A}^{(-\alpha)}$ now has one sort A for the elements of A and the complement of A contains infinitely many auxiliary elements. For each $i \in I$, there is also an $a_i + 2$-ary relation Q_i assigning to each a_i-tuple $\bar{u}$ an infinite set $U_{i,\bar{u}} = \{x : Q_i(\bar{u}, x, x)\}$, and a 2-ary relation $\bar{R}_{i,\bar{u}} = \{(x, y) : Q_i(\bar{u}, x, y)\}$ on $U_{i,\bar{u}}$. The sets $U_{i,\bar{u}}$ are infinite and partition the complement of A. For each i and $\bar{u} \in A^{a_i}$, $(U_{i,\bar{u}}; \bar{R}_{i,\bar{u}})$ is a structure $\mathcal{U}_{i,\bar{u}}$ which is isomorphic to either $\mathcal{B}_0$ or $\mathcal{B}_1$ depending on whether $\mathcal{A} \models R_i(\bar{u})$ or not.

Using Ash and Knight's theorem mentioned above, one can then prove the following lemma:

Lemma 6.2 (See Lemma 5.5 of [GHK+05])**.** *Let α be a computable successor ordinal and let $\mathcal{A}$ be a structure. Then $\mathcal{A}$ has a Δ^0_α copy if and only if $\mathcal{A}^{(-\alpha)}$ has a computable copy.*

We can also prove a similar fact about the back-and-forth trees of a structure.

Lemma 6.3. *Let $\mathcal{A}$ be a structure and suppose that $\mathcal{T}(\mathcal{A})$ has a Δ^0_α copy. Then $\mathcal{T}(\mathcal{A}^{(-\alpha)})$ has a computable copy.*

Proof. This is essentially the same proof as the lemma above. Construct a computable sequence of structures $\mathcal{C}_{\bar{\tau},i}$, with $\bar{\tau}$ a tuple of nodes from the same path in $\mathcal{T}(\mathcal{A})$ of length a_i, such that $\mathcal{C}_{\bar{\tau},i} \cong \mathcal{B}_0$ if $\bar{\tau} = \langle \tau_1, \ldots, \tau_{a_i} \rangle$ satisfies R_i in $\mathcal{T}(\mathcal{A})$ and $\mathcal{C}_{\bar{\tau},i} \cong \mathcal{B}_1$ otherwise.

The underlying tree of $\mathcal{T}(\mathcal{A}^{(-\alpha)})$ will be the tree of tuples $\langle b_1, \ldots, b_n \rangle$ where

1. each b_j is either a node τ in $\mathcal{T}(\mathcal{A})$ or is of form $\langle \tau_j^1, \ldots, \tau_j^{a_i}, i, n \rangle$ for some i and $n \in \mathcal{C}$,
2. all of the τ come from the same path in $\mathcal{T}(\mathcal{A})$,
3. after deleting from $\langle b_1, \ldots, b_n \rangle$ all of the entries of the form $\langle \tau_j^1, \ldots, \tau_j^{a_i}, i, n \rangle$, the remaining entries (which are nodes $\tau \in \mathcal{T}(\mathcal{A})$) form an initial segment of a path in $\mathcal{T}(\mathcal{A})$.

Now, given a tuple $\langle \sigma_1, ..., \sigma_k \rangle$ of nodes within the same path in $\mathcal{T}(\mathcal{A}^{(-\alpha)})$, let Q_i hold of it if $\langle last(\sigma_1), ..., last(\sigma_k) \rangle$ is of the form $\langle \bar{\tau}, b_1, b_2 \rangle$, where $\bar{\tau} \in \mathcal{T}(\mathcal{A})$, b_1 and b_2 are of the form $(\bar{\tau}, i, n_1)$ and $(\bar{\tau}, i, n_2)$, and (n_1, n_2) satisfies the binary relation in $\mathcal{C}_{\bar{\tau},i}$. □

Proof of Corollary 1.11. We may assume that α is a successor ordinal. Relativize Theorem 1.10 to $0^{(\alpha)}$ to obtain a structure $\mathcal{A}$ be such that $\mathcal{T}(\mathcal{A})$ has a $0^{(\alpha)}$-computable copy but $\mathcal{A}$ has no $0^{(\alpha)}$-computable copy. Let $\mathcal{B}$ be $\mathcal{A}^{(-\alpha)}$. Then by Lemma 6.3 $\mathcal{T}(\mathcal{B})$ has a computable copy, but by Lemma 6.2 $\mathcal{B}$ has no $0^{(\alpha)}$-computable copy. □

7 Linear orders

Friedman and Stanley [FS89b] proved that linear orders are Borel complete. Recall that this means that for each fixed language, there is a Borel operator Φ that takes a structure $\mathcal{A}$ to a linear order $\Phi(\mathcal{A})$ such that

$$\mathcal{A} \cong \mathcal{B} \iff \Phi(\mathcal{A}) \cong \Phi(\mathcal{B}).$$

Their operator Φ is in fact computable and can be defined as the composition of the tree-of-tuples operator $\mathcal{T}$ and the following operator $\mathcal{L}$.

Definition 7.1. Let T be a labeled tree. Define $\mathcal{L}(T)$ recursively as follows: $\mathcal{L}(T)$ is the shuffle sum of $\mathbb{Q} + \ell(\sigma) + 2 + \mathbb{Q} + \mathcal{L}(T_\sigma)$ for all children σ of the root node, where T_σ is the subtree of T below σ and $\ell(\sigma)$ is the (integer code for) the label of σ.

So the Friedman-Stanley operator is $\Phi = \mathcal{L} \circ \mathcal{T}$.

Corollary 7.2. *For each computable ordinal α, there is a structure $\mathcal{A}$ such that $\Phi(\mathcal{A})$ is computable but $\mathcal{A}$ has no Δ^0_α copy.*

Proof. Note that one can recover the label tree T from the triple jump of $\mathcal{L}(T)$, and then apply Theorem 1.11 to $\alpha + 3$. □

References

[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.

[Ers96] Y. L. Ershov. *Definability and computability.* Siberian School of Algebra and Logic. Consultants Bureau, New York, 1996.

[FS89a] Harvey Friedman and Lee Stanley. A Borel reducibility theory for classes of countable structures. *J. Symbolic Logic*, 54(3):894–914, 1989.

[FS89b] Harvey Friedman and Lee Stanley. A Borel reducibility theory for classes of countable structures. *J. Symbolic Logic*, 54(3):894–914, 1989.

[GHK+05] Sergey Goncharov, Valentina Harizanov, Julia Knight, Charles McCoy, Russell Miller, and Reed Solomon. Enumerations in computable structure theory. *Ann. Pure Appl. Logic*, 136(3):219–246, 2005.

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

[Kal09] Iskander Kalimullin. Enumeration degrees and enumerability of families. *J. Logic Comput.*, 19(1):151–158, 2009.

[Kal12] Iskander Kalimullin. Algorithmic reducibilities of algebraic structures. *J. Logic Comput.*, 22(4):831–843, 2012.

[Kni86] Julia F. Knight. Degrees coded in jumps of orderings. *J. Symbolic Logic*, 51(4):1034–1042, 1986.

[KP09] I. Sh. Kalimullin and V. G. Puzarenko. Reducibility on families. *Algebra Logika*, 48(1):31–53, 150, 152, 2009.

[KSV] J. Knight, A. Soskova, and S. Vatev. Coding in graphs and linear orderings. preprint.

[Mon13a] A. Montalbán. A fixed point for the jump operator on structures. *Journal of Symbolic Logic*, 78(2):425–438, 2013.

[Mon13b] Antonio Montalbán. A fixed point for the jump operator on structures. *J. Symbolic Logic*, 78(2):425–438, 2013.

[Mon14] A. Montalbán. Computability theoretic classifications for classes of structures. In S. Y. Jang, Y. R. Kim, D.-W. Lee, and I. Yie, editors, *Proceedings of the International Congress of Mathematicians (Seoul 2014). Vol. II*, pages 79–101. Kyung Moon Sa Co., Seoul, 2014.

[MonP1] Antonio Montalbán. Computable structure theory: Within the arithmetic. In preparation, P1.

[MPSS18] R. Miller, B. Poonen, H. Schoutens, and A. Shlapentokh. A computable functor from graphs to fields. *Journal of Symbolic Logic*, 83(1):326–348, 2018.