PUNCTUAL CATEGORICITY AND UNIVERSALITY

ROD DOWNEY, NOAM GREENBERG, ALEXANDER MELNIKOV, KENG MENG NG, AND DANIEL TURETSKY

ABSTRACT. We describe punctual categoricity in several natural classes, including binary relational structures and mono-unary functional structures. We prove that every punctually categorical structure in a finite unary language is PA($0'$)-categorical, and we show that this upper bound it tight. We also construct an example of a punctually categorical structure whose degree of categoricity is $0''$. We also prove that, with a bit of work, the latter result can be pushed beyond Δ^1_1, thus showing that punctually categorical structures can posses arbitrarily complex automorphism orbits.

As a consequence, it follows that binary relational structures and unary structures are not universal with respect to primitive recursive interpretations; equivalently, in these classes every rich enough interpretation technique must necessarily involve unbounded existential quantification or infinite disjunction. In contrast, it is well-known that both classes are universal for Turing computability.

1. INTRODUCTION

It is well-known that decidability of the Word Problem in a finitely generated group does not depend on the choice of its presentation. Indeed, every two such presentations are computably isomorphic, in particular preserving computability (or non-computability) of the Word Problem. Similarly, Cantor's back-and-forth proof shows that any two computable copies of $(\mathbb{Q}, <)$ are computably isomorphic. Also, it is well-known that any structure $\mathcal{A}$ in a fixed finite language can be turned into a graph $G(\mathcal{A})$ or into a unary functional structure $U(\mathcal{A})$ such that both $G(\mathcal{A})$ and $U(\mathcal{A})$ possess the same model-theoretic and decidability properties as $\mathcal{A}$. Some of these results are so basic and so "obviously computable" that they are often used without explicit reference.

But what happens if we put some resource bounds on our effective procedures? Which of these "algorithms" can be transformed into more feasible ones, and which are *provably* inefficient? Is there any correlation between feasible computability on infinite algebraic structures and definability upon these structures in some natural language? Can we turn any structure into, say, a unary structure preserving most "online computable" features of the structure?

In this article we use the recently suggested *punctual structure theory* to systematically investigate these and similar questions.

1.1. The punctual framework.

Kalimullin, Melnikov and Ng [KMN17] have initiated a systematic study which is focused on eliminating unbounded search from proofs and processes in algebra and infinite combinatorics. The main underlying

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

abstraction in the new framework is the old classical notion of a primitive recursive algorithm which can be traced back to Kronecker. Informally, an algorithm is primitive recursive if every loop and search operator in the algorithm has a precomputed bound. Although a primitive recursive algorithm does not have to be computationally feasible, it serves as a useful abstraction which unites most common complexity classes of interest. In fact, as discussed in [KMN17, BDKM19], very often eliminating unbounded search is the crucial step in turning a general Turing computable algebraic procedure into, say, a polynomial time or a polylogspace one; see, e.g., [Gri90, CDRU09, CR92, CR98]. A recent and non-trivial illustration of this phenomenon is the recent solution [BHTK$^+$19] to a problem of Khouissainov and Nerode on the characterisation of automatic structures ([KN08], Question 4.9). The key step in the proof in [BHTK$^+$19] is a simpler argument for primitive recursive structures; with some extra work it is then pushed to the extremely narrow class of automatic structures.

Another useful role of primitive recursion is in proving that no feasible procedure is possible at all. Indeed, it is often easiest to argue that a primitive recursive procedure fails to exist, let alone a polynomial or exponential time procedure; see, e.g., [CR92, CR98, KMN17]. In such proofs one can typically diagonalise even against all total (Turing) computable procedures, i.e., against those procedures which eventually halt [Kie98, KPT94]. Thus with all its generality, primitive recursion could be even a bit too narrow for such proofs; nonetheless, every total Turing computable function can be viewed as a function primitive recursive relative to some (functional) oracle. Therefore, the above-mentioned totality phenomenon is still within the reach of this framework; see the recent paper by Kalimullin, Melnikov and Montalbán for more on totality, relativisation, and definability in primitive recursive algebra [KMM19]. We cite the recent surveys [BDKM19, Mel17, DMN] for a detailed exposition of the framework and its connections with computable structure theory, Weihrauch reducibility, infinite games on structures, incremental reducibility in computer science, complexity of real functions, and feasible combinatorics.

1.2. **Categoricity and universality.** One of the central definitions of the new framework is that of a *punctual presentation of a structure*; this is an isomorphic copy of a given countably infinite structure whose domain is $\mathbb{N}$ and the operations and relations are primitive recursive[1]. The structure is "punctual" in the sense that it reveals itself without unbounded delay. In several broad classes, including linear orders and torsion-free abelian groups, one can show that every Turing computable structure has a polynomial time presentation [Gri90, CDRU09, KMN17]; for these classes it is sufficient to build a punctual copy and observe that it is actually polynomial time.

The natural morphisms in the category of punctual structures are the isomorphisms f for which both f and f^{-1} are primitive recursive. We call such isomorphisms *punctual*.

Definition 1.1. A structure is *punctually categorical* if it has a unique punctual presentation up to punctual isomorphism.

[1]Although the definition can be pushed to infinite languages, we assume that the language of a structure is finite. If the structure itself is finite then the domain of its punctual presentation will be an initial segment of $\mathbb{N}$.

The notion above is the most natural primitive recursive analogue of the notion of computable categoricity, which is central to computable structure theory [AK00, EG00]. Recall that a structure is computably categorcial if it has a unique (Turing) computable presentation up to (Turing) computable isomorphism. In computable structure theory, the study of computable categoricity and its generalisations revealed deep connections between algebraic, algorithmic, and syntactical properties of structures. For instance, there are a large number of purely algebraic characterisations of computably categorical members of standard classes [AK00]. It is also well-known that a structure is relatively computably categorical if, and only if, there exists a computably enumerable list of existential first-order formulae which describe automorphism orbits of tuples of the structure (see for example [DHK03]).

The lack of reasonable classification of computably categorical structures [DKL$^+$15] shows that computable categoricity and definability can vastly differ in many standard classes including undirected graphs (folklore), unary functional structures (folklore), two-step nilpotent groups [HKSS02], lattices [BFKMn17, HKSS02], and notably fields [MPSS18]. In all these classes, computability, algebra, and definability can significantly diverge, and among other insights this leads to intricate counterexamples disproving natural conjectures, such as Goncharov's famous dimension two examples [Gon81, Gon80]. Although such results were initially designed to defeat regularity and definability, there is a certain general methodology behind such proofs which itself heavily relies on definability. For example, Goncharov's original dimension two counterexample [Gon80] was designed for families of subsets of $\mathbb{N}$. To obtain similar examples for unary structures or two-step nilpotent groups, it is sufficient to *effectively turn* any such family into a structure from the respective class which *effectively encodes* the family in a way that preserves computable dimension and categoricity [GMR89]. In [HKSS02] this idea was made explicit and more precise, and in [HTMMM17] it was shown that the two most natural general definitions of effective interpretations, one via Turing effective functors and the other using definability, are in fact equivalent.

Apart from various definability techniques, some of which are quite intricate, such results provide us with proofs that some natural classes are *Turing universal*, and some are not. Informally, a class is Turing universal if any countable structure can be Turing computably turned into a structure from this class preserving most decidability-theoretic properties of interest; see [HTMMM17] for technical details. The majority of such proofs in the literature rely on indirect Δ^0_0-definability, meaning that both a relation and its complement are $\exists$-definable. It is rather natural to ask whether such definitions can be turned into proper direct Δ^0_1-definability, meaning that all quantifiers have to be bounded in some reasonable predetermined way, and so that the bound does not vary from a presentation to a presentation. Equivalently, if F is a functor translating a structure A into a structure $F(A)$, can we avoid searching for witnesses throughout the entire structure $F(A)$ to reconstruct A? Is there a more feasible definition of A within $F(A)$? Can the functor F itself be made more feasible, and made to preserve feasible algorithmic features of structures? For instance, which of the standard Turing universal classes remain *punctually universal*?

The formal definition of punctual universality can be found in [HTMMM17]; we omit it. It will be sufficient to know that it is obtained from the notion of Turing universality by replacing Turing functionals with primitive recursive functionals throughout the definition. In this paper it is only important to know that every punctually universal class must be Turing universal, because a primitive recursive functional is obviously a Turing functional too. So, for instance, the transformations witnessing punctual universality preserve not only punctual categoricity but also computable categoricity and its generalisation Δ^0_α-categoricity (see [AK00]).

A natural example of a punctually universal class is the class of structures in the language of one binary functional symbol [DHTK$^+$]. It is well-known that graphs are Turing universal. It has recently been discovered that every punctually categorical graph becomes automorphically trivial after fixing finitely many constants [DHTK$^+$]. In particular, every punctually categorical graph must be (relatively) computably categorical. Remarkably, Kalimulliun, Malnikov and Ng [KMN17] constructed an example of a punctually categorical structure which is *not* computably categorical. Since punctual universality must respect both punctual and computable categoricity, it follows that *graphs are not punctually universal*. Indeed, if they were punctually universal then we could punctually interpret the example from Kalimulliun, Malnikov and Ng [KMN17] and obtain a punctually categorical graph which is not computably categorical, contradicting the above-mentioned description of punctually categorical graphs [DHTK$^+$]. Recently Kalimullin and Miller [KM19] have obtained a purely algebraic description of punctually categorical fields. Similarly to the case of graphs, the description entails that the Turing universal class of fields [MPSS18] is not punctually universal.

These results show that unbounded $\exists$-quantification or the use of infinite disjunction (equivalently, unbounded μ-operator) is *intrinsic* to any powerful enough coding technique in these classes. Such results give an explicit correlation between punctual universality, describing punctual categoricity in natural classes, and pushing the technical boundaries of the topic by producing highly counter-intuitive examples. The latter two themes are of independent interest as well.

In this article we continue the systematic investigation of punctual categoricity in natural classes, with applications to punctual universality. Before we state our results, we note that our proofs tend to be rather combinatorially involved with Theorem 1.3 being perhaps the only pleasant exception. We suspect that this complexity cannot be avoided. To sort out some of this combinatorics we develop a technical framework which will be discussed in Section 2.

1.3. **The results.** The first main result of the paper extends the main result from [DHTK$^+$] to arbitrary structures in the languadge of finitely many at most binary relations.

Theorem 1.2. *Every punctually categorical structure with at most binary relational symbols is automorphically trivial.*

This solves a problem left open in [DHTK$^+$]; see also [BDKM19]. The proof of the theorem is not merely a generalisation of the proof from [DHTK$^+$] since it relies on a new strategy; it is also combinatorially more intricate. As a consequence, we obtain that the class of all structures in any finite language containing at most binary relational symbols is not punctually universal. We note that this corollary does not need to rely on the analysis of punctual categoricity. In fact, the more direct argument in [KMM19] covers the more general case of arbitrary relational structures. However, the methods in [KMM19] do not seem to help with the description of punctual categorcity in the class.

In contrast with binary relational structures, a punctually categorical structure with only one unary functional symbol (a mono-unary structure) does not have to be automorphically trivial. For example, a punctually categorical mono-unary structure may consist of infinitely many loops of some fixed size. Another example of a punctually categorical mono-unary structure is an infinite star; this is a unary structure (X, u) in which for some $y \in X$ we have $(\forall z \neq y)(u(z) = y)$. (Note that there are exactly two isomorphism types described by the property above, depending on $u(y)$.) Interestingly, the two examples above essentially describe punctual categoricity for such structures.

Theorem 1.3. *Suppose X is an infinite structure with only one unary functional symbol. Then X is punctually categorical if and only if there is a finite subset F of X such that either:*

1. *$X \setminus F$ is a disjoint union of loops of identical size,*
2. *$X \setminus F$ is an infinite star.*

In both 1. and 2. of the theorem above, the structures are (relatively) computably categorical. Although it is not known whether the class is Turing universal (and it perhaps is not), from the perspective of Turing computability it is at least no simpler than the rich class of trees. However, it turns out that in the punctual world mono-unary structures are rather tame.

The case of several unary functional symbols is significantly harder to grasp. We say that a structure is $\mathrm{PA}(0')$-categorical if, for every pair of its computable copies and every $X \in \mathrm{PA}(0')$, there is an isomorphism between the copies computable from X. In particular, there is always a low over $0'$ isomorphism between any two copies of the structure. This notion is not really new. Hirschfeldt and Khoussainov (see, e.g., [Hir17]) observed that every locally finite connected graph is $\mathrm{PA}(0')$-categorical. We note that, building on a work of Gromov [Gro07], Melnikov and Nies [MN13] obtained a similar bound for compact separable Polish spaces. We prove:

Theorem 1.4. *Every punctually categorical unary structure is* $\mathrm{PA}(0')$*-categorical*[2].

The proof of Theorem 1.4 goes through several cases and uses various strategies, most of which are new. The proof of the theorem shows that the orbit of every tuple in such a structure must be finite. It is natural to ask whether the theorem above can be pushed to a purely algebraic description or at least to (relative) Δ^0_2-categoricity. However, it turns out that the upper bound established in the theorem above is tight.

Theorem 1.5. *There exists a punctually categorical unary structure for which the categoricity spectrum is precisely the* $\mathrm{PA}(0')$ *degrees.*

The proof of Theorem 1.5 is of some independent interest. For instance, in the proof we will introduce a new strategy of building a punctually categorical structure.

---

[2]Without loss of generality we could also allow unary relations in the language. A unary relation can be imitated by a unary function as follows: $P(x) \iff u(x) = x$; we could set $u(x)$ equal to some special constant if $\neg P(x)$. Note that this definition is quantifier-free and thus punctual. We also note that the proof shows that such structures are "relatively $\mathrm{PA}(\Delta^0_2)$-categorical"; we leave the fromal clarification of this notion to the reader.

In the Turing computable world, the class of structures with just two unary functional symbols is already universal. Perhaps the same could hold in the world of punctual structures. Perhaps, *every* punctually categorical structure, unary or not, could be PA($0'$)-categorical. In fact, it seemed that finiteness of automorphism orbits must be a characteristic property shared among all punctually categorical structures. Nonetheless, using a novel and rather involved machinery we prove:

Theorem 1.6. *There is a punctually categorical structure whose degree of (Turing) categoricity is* $0''$.

Theorems 1.4 and 1.6 imply:

Corollary 1.7. *The class of all structures in any finite language containing at most unary symbols is not punctually universal.*

In contrast with relational structures, we do not know any direct proof of non-universality of unary structures which would not filter through Theorems 1.5 and 1.6; we leave this as an open problem. In the proof of Theorem 1.6 we develop a relatively complicated apparatus of macro-labels which we also combine with the new "preventing" strategy. The significance of these new techniques is that they allow us to produce complex enough punctually categorial structures which have infinite automorphism orbits (in contrast with, e.g., examples in [KMN17] or in Theorem 1.5). Once these two techniques are described and verified, it is actually not that hard to push the proof of Theorem 1.6 beyond any computable ordinal α:

Theorem 1.8. *There is a punctually categorical structure which is not* Δ^1_1-*categorical.*

The theorem will reappear as Corollary 8.3 of a more general Theorem 8.1 which essentially turns any Turing computable structure into a punctually categorical one. These results show that punctually categorical structures can be arbitrarily complicated. It also illustrates the power of the macro-label technique which we expect will find further applications. Theorem 1.8 solves a problem left open in [BDKM19]. Also, Theorem 1.8 will likely be useful in showing that certain classes of structures are not punctually universal, similarly to how Theorem 1.6 was used to establish punctual non-universality of unary structures.

2. A general framework

Throughout this paper, we will be showing results of the form "If a structure $\mathcal{A}$ is punctually categorical, then it has property X." We will argue these by contrapositive; under the assumption that $\mathcal{A}$ does not have property X, we will build a punctual structure $\mathcal{B}$ which is isomorphic to $\mathcal{A}$ (often by a computable isomorphism) but not bi-primitive recursively so.

We present here a general framework for most of these arguments (but not all; see for example Lemmas 3.2 and 3.3). A construction will have two phases: a *diagonalization* phase and a *recovery* phase, which it will alternate between. The diagonalization phase will be lengthy, such that we expect no primitive recursive bound on the lengths of each phase, and a priori there may even be a diagonalization phase which never ends. The phases will be designed to have the following properties:

- In all cases, we construct a punctual structure $\mathcal{B}$.

- If there is a diagonalization phase which never ends, then $\mathcal{B} \ncong \mathcal{A}$.
- If every diagonalization phase eventually ends (and thus there are infinitely many recovery phases), then $\mathcal{B} \cong \mathcal{A}$.

Each diagonalization phase will be targeted for a particular pair of primitive recursive functions (p, q), as we aim to show that either $p : \mathcal{A} \to \mathcal{B}$ is not an embedding or at least q is not its inverse. Note that as primitive recursive functions, p and q are total. So if p is not an embedding, we see proof at some finite stage: either elements x, y with $x \neq y$ but $p(x) = p(y)$, or some atomic formula θ and some $\bar{x}$ with $\mathcal{A} \models \theta(\bar{x}) \iff \mathcal{B} \not\models \theta(p(\bar{x}))$. Similarly, if q is not the inverse of p, we will see some x with $p(q(x)) \neq x$. Once this occurs, (p, q) is forever defeated; there is no chance our construction might inadvertently rescue the pair.

A diagonalization phase will continue until we see proof that the targeted pair is not an isomorphism. If the diagonalization phase never ends, then by construction there can be no isomorphism from $\mathcal{A}$ to $\mathcal{B}$, and so in particular either p is not an embedding, or it is not surjective and thus q is not its inverse. Thus we will eventually see the desired proof and end the diagonalization strategy. So our construction will have infinitely many recovery phases, meaning $\mathcal{B} \cong \mathcal{A}$. We will also successfully diagonalize against every pair, and so we will have shown that $\mathcal{A}$ is not punctually categorical.

3. At most binary relations

Recall that a structure is *automorphically trivial* if there is a finite subset F such that every permutation of the structure which fixes F pointwise is an automorphism. An automorphically trivial structure is clearly punctually categorical: nonuniformly map the finite set F, and then extend to the rest of the structure via any bi-primitive recursive permutation.

Theorem 3.1. *Every punctually categorical structure with only binary and unary relational symbols is automorphically trivial.*

Proof. Consider a punctually categorical structure $\mathcal{A}$ whose signature consists of finitely many binary and unary relations. We show such a structure must be automorphically trivial.

Consider every possible 2-element atomic diagram in our signature. Each is a color. We have an inverse operation on colors, so that $c(x, y) = a \iff c(y, x) = a^{-1}$. Some colors are their own inverses. Note that $c(\cdot, \cdot)$ is primitive recursive.

Lemma 3.2. *If such a structure $\mathcal{A}$ is punctually categorical, then for every $x \in \mathcal{A}$ and every color a such that there are infinitely many y with $c(x, y) = a$, there is a primitive recursive function f such that for all n, there are at least n elements y with $c(x, y) = a$ and $y < f(n)$.*

Proof. Suppose not. Then fix an $x \in \mathcal{A}$ and a color a forming a counterexample. That is, if $N^{\mathcal{A}}(x) = \{y : c(x, y) = a\}$, and $N^{\mathcal{A}}_s(x)$ is the natural stage-based approximation, then for any primitive recursive function f, there are infinitely many s such that $N^{\mathcal{A}}_s(x) = N^{\mathcal{A}}_{f(s)}(x)$. By a primitive recursive renumbering of stages, we assume that $\mathcal{A}$ is fully defined on $[0, s)$ by stage s.

We construct two structures $\mathcal{B}_0$ and $\mathcal{B}_1$, along with primitive recursive maps $\varphi_0, \varphi_1 : \omega \to \mathcal{A}$, such that $\varphi_i : \mathcal{B}_i \to \mathcal{A}$ is an isomorphism. Indeed, $\mathcal{B}_i$ will simply be defined to be the pull-back of $\mathcal{A}$ along φ_i. To ensure that each $\mathcal{B}_i$ is primitive recursive, we will maintain that the domain of φ_i is either $[0, s-1)$ or $[0, s)$ at stage s, while the range is contained in $[0, s)$. For at least one of the i, and possibly both, we will have that $\mathrm{dom}(\varphi_i) = [0, s)$ at stage s.

We must diagonalize against each pair (p_0, p_1) of primitive recursive functions, showing that at least one of the following holds:

- $p_0 : \mathcal{B}_0 \to \mathcal{B}_1$ is not an isomorphism; or
- p_0 and p_1 are not inverses.

We consider each such pair in some effective order, addressing each in turn. While we are working for a pair (p_0, p_1), we are always watching for proof that we have succeeded. That is, if at stage s there is a $z < s$ such that $p_0(z)$ and $p_1(p_0(z))$ have both converged, but $p_1(p_0(z)) \neq z$, then we have proof that p_0 and p_1 are not inverses. If there are $w, z < s-1$ such that $p_0(w), p_0(z) < s-1$ have converged by stage s, but $c(\varphi_0(w), \varphi_0(z)) \neq c(\varphi_1(p_0(w)), \varphi_1(p_0(z)))$, then we have proof that p_0 is not an isomorphism. Once we have proof, we discard the pair (p_0, p_1) and move on to the next pair.

When we discard a pair at stage s, it may be that the domain of one of the φ_i at stage s is $[0, s-1)$. So there is some element $u < s$ with $u \notin \mathrm{range}(\varphi_i)$. We extend φ_i at stage $s+1$ with $\varphi_i(s-1) = u$, $\varphi_i(s) = s$. In this way we are certain that when we begin considering the next pair at stage $s+1$, we have $\mathrm{dom}(\varphi_i) = \mathrm{range}(\varphi_i) = [0, s]$.

Before we even consider the first pair of primitive recursive functions, we begin by simply defining $\varphi_i(s) = s$ at every stage $s+1$ (for each $i < 2$). This continues until we reach a stage s with $x < s$, and thus $x \in \mathrm{range}(\varphi_i)$. Once this occurs, we begin considering the first pair of primitive recursive functions.

Let $z_0 = \varphi_0^{-1}(x)$. Of course, inspection will reveal that $z_0 = x$, but it is convenient to have a different name for the element to distinguish when we are considering it as an element of $\mathcal{A}$ versus as an element of $\mathcal{B}_0$.

Our strategy for the pair (p_0, p_1). By assumption, at stage s, we have $\varphi_i : [0, s) \to [0, s)$ a bijection, for each $i < 2$. If, by stage s, $p_0(z_0)$ has not converged, or it has converged to a value greater than or equal to s, we simply extend φ_i at stage $s+1$ by $\varphi_i(s) = s$, for both $i < 2$.

If instead $p_0(z_0) < s$ has converged by stage s, we define $z_1 = p_0(z_0)$. Note that there is no reason to believe $\varphi_1(z_1) = x$. The element $\varphi_1^{-1}(x)$ will have no special role in our construction.

For $i < 2$, define $N_s^i = \{y \in \mathrm{dom}(\varphi_{i,s}) : c(\varphi_i(z_i), \varphi_i(y)) = a\}$. Since $\mathcal{B}_i$ is defined by the pullback of φ_i, this is the set of all y such that $c(z_i, y) = a$ in $\mathcal{B}_i$. We wish to engineer a situation in which the following holds at stage s:

- $|N_s^0| \neq |N_s^1|$; and
- If $j_0, j_1 < 2$ are such that $|N_s^{j_1}| > |N_s^{j_0}|$, then:
  - $\mathrm{dom}(\varphi_{j_1}) = [0, s)$; and
  - If $\mathrm{dom}(\varphi_{j_0}) = [0, s-1)$, and $u < s$ is the unique element not in the $\mathrm{range}(\varphi_{j_0})$, then $c(\varphi_{j_0}(z_{j_0}), u) = a$.

We henceforth refer to this situation as "the desired state". Until we are in the desired state, at every stage $s+1$, we extend each φ_i by defining $\varphi_i(s) = s$, unless choosing not to extend φ_0 would put us into the desired state at stage $s+1$. In that case, we extend φ_1 but not φ_0. So until we are in the desired state, we have $\mathrm{dom}(\varphi_{i,s}) = [0, s)$ for both $i < 2$.

Since there are infinitely many y with $c(x,y)=a$, let y_0 be the least such y which is greater than the stage s_0 at which we began considering the pair (p_0,p_1). At stage $s=y_0$, we may already be in the desired state. If not (meaning $|N_s^0|=|N_s^1|$), there are two possibilities. If $c(\varphi_1(z_1),y_0)\neq a$, then extending $\varphi_i(s)=s$ for both i will put us into the desired state with $(j_0,j_1)=(1,0)$ and $\mathrm{dom}(\varphi_{j_0})=[0,s)$. If $c(\varphi_1(z_1),y_0)=a$, then extending $\varphi_1(s)=s$ and not extending φ_0 will put us in the desired state with $(j_0,j_1)=(0,1)$ and $u=y_0$.

Thus we will eventually enter the desired state (or see proof that we have defeated the pair (p_0,p_1)). Once in the desired state, the remainder of our strategy is to simply maintain the desired state (possibly interchanging the j_0 and j_1 as we do so). We explain how this is done.

If $|N_s^{j_1}|-|N_s^{j_0}|>1$, then at stage $s+1$ we can extend each φ_i by letting r be the least element not in its domain (so $r\in\{s-1,s\}$) and defining $\varphi_i(r)=s$. Since $|N^{j_0}|$ can increase by at most one element between stages s and $s+1$ (specifically, the element r), while $N_s^{j_1}\subseteq N_{s+1}^{j_1}$, we maintain the desired state.

If $|N_s^{j_1}|-|N_s^{j_0}|=1$ and at least one of $c(\varphi_{j_0}(z_{j_0}),s)\neq a$ or $c(\varphi_{j_1}(z_{j_1}),s)=a$ holds, then we can again extend each φ_i by $\varphi_i(r)=s$. If we have $c(\varphi_{j_0}(z_{j_0}),s)\neq a$, we will have $N_s^{j_0}=N_{s+1}^{j_0}$, while $N_s^{j_1}\subseteq N_{s+1}^{j_0}$, and so the desired state is maintained. If we have $c(\varphi_{j_1}(z_{j_1}),s)=a$, we will have $|N_{s+1}^{j_1}\setminus N_s^{j_1}|=1$, while $|N_{s+1}^{j_0}\setminus N_s^{j_0}|\leq 1$, and so again the desired state is maintained.

If $|N_s^{j_1}|-|N_s^{j_0}|=1$, $c(\varphi_{j_0}(z_{j_0}),s)=a$ and $c(\varphi_{j_1}(z_{j_1}),s)\neq a$, then our behavior depends on whether or not $\mathrm{dom}(\varphi_{j_0,s})=[0,s)$. If so, then we can choose not to extend φ_{j_0}, defining $\varphi_{j_0,s+1}=\varphi_{j_0,s}$ while extending φ_{j_1} by $\varphi_{j_1}(s)=s$. Then we have maintained the desired state, now with $u=s$.

If $|N_s^{j_1}|-|N_s^{j_0}|=1$, $c(\varphi_{j_0}(z_{j_0}),s)=a$, $c(\varphi_{j_1}(z_{j_1}),s)\neq a$ and $\mathrm{dom}(\varphi_{j_0,s})=[0,s-1)$, then we can extend by $\varphi_{j_0}(s-1)=u$, $\varphi_{j_0}(s)=s$ and $\varphi_{j_1}(s)=s$. Now we have $|N_{s+1}^{j_0}\setminus N_s^{j_0}|=2$, while $N_{s+1}^{j_1}=N_s^{j_1}$. So we have maintained the desired state, albeit by interchanging the roles of j_0 and j_1.

Next, we argue that if we maintain the desired state indefinitely, we will eventually see proof that we have defeated the pair (p_0,p_1). Define the function f such that $f(s)$ is the least $t>s$ such that p_0 and p_1 have both converged on all of $[0,s)$ by stage t, and for all $w\in[0,s)$, we have $p_0(w)<t$ and $p_1(w)<t$. Note that f is primitive recursive, and thus so is $f\circ f$. Then, as previously observed, there are infinitely many s such that $N_s^{\mathcal{A}}=N_{f(f(s))}^{\mathcal{A}}$. Fix such an s_1 after the stage at which we begin maintaining the desired state for the pair (p_0,p_1).

Suppose we have not seen the desired proof by stage s_1, and thus are still maintaining the desired state at stage s_1. There are two cases. In case (1), we are maintaining the desired state with $j_0=0$. An examination of our strategy for maintaining the desired state will show that since there is no $s\in[s_1,f(f(s_1))$ with $c(x,s)=a$, $N_{s_1}^{j_0}=N_{f(s_1)}^{j_0}$. But by assumption, $|N_{s_1}^{j_1}|>|N_{s_1}^{j_0}|$, and p_1 converges on all of $N_{s_1}^{j_1}$ by stage $f(s_1)$. Thus, by counting, either p_1 fails to be injective, or $p_1(z_1)\neq z_0$, or there is a $y\in N_{s_1}^{j_1}=N_{s_1}^1$ such that $c(\varphi_1(z_1),\varphi_1(y))=a$ and $c(\varphi_0(p_1(z_1)),\varphi_0(p_1(y)))\neq a$, thus witnessing that p_1 is not an isomorphism or is at least not the inverse of p_0.

In case (2), we are instead maintaining the desired state with $j_0=1$. There are two subcases. We know that we are maintaining the desired state at stage $f(s_1)$. Case (2a) is that at stage $f(s_1)$ we have not interchanged the roles of j_0 and j_1.

Then we have $|N^{j_0}_{s_1}| \leq |N^{j_0}_{f(s_1)}| < |N^{j_1}_{f(s_1)}| = |N^{j_1}_{s_1}|$. This final equality is again by the fact that $N^{\mathcal{A}}_{s_1} = N^{\mathcal{A}}_{f(s_1)}$ and an examination of our strategy for maintaining the desired state. But by assumption, p_0 converges on all of $N^{j_1}_{s_1}$ by stage $f(s_1)$. Thus, by counting, either p_0 fails to be injective, or there is a $y \in N^{j_1}_{s_1} = N^0_{s_1}$ such that $c(\varphi_0(z_0), \varphi_0(y)) = a$ and $c(\varphi_1(p_0(z_0)), \varphi_1(p_0(y))) \neq a$, thus witnessing that p_0 is not an isomorphism.

Case (2b) is that we are maintaining the desired state at stage s_1 with $j_0 = 1$ and at stage $f(s_1)$ with $j_0 = 0$. Then, as in Case 1, by stage $f(f(s_1))$ we still have $j_0 = 0$. The argument now proceeds as in Case (1), using stages $f(s_1)$ and $f(f(s_1))$. □

Lemma 3.3. *If such a structure $\mathcal{A}$ is punctually categorical, then the coloring is stable, in that* $(\forall x \in \mathcal{A})(\exists a)(\forall^\infty y \in \mathcal{A})\, c(x, y) = a$.

Proof. Suppose not. Then fix an $x \in \mathcal{A}$ forming a counterexample. Since there are only finitely many colors, by pigeon hole there must be distinct colors a and b such that $\exists^\infty y\, c(x, y) = a$ and $\exists^\infty y\, c(x, y) = b$. Fix an infinite primitive recursive set D such that the principal function of D grows faster than any primitive recursive function. By a primitive recursive renumbering of stages, we assume that both $\mathcal{A}$ and D are fully defined on $[0, s)$ by stage s.

By lemma 3.2, there is a primitive recursive f such that for all n, there are at least $n+1$ distinct $y \in [0, f(n))$ such that $c(x, y) = a$. We may assume f is strictly increasing, and thus has a primitive recursive range. We will build a punctual structure $\mathcal{B} \cong \mathcal{A}$ violating lemma 3.2 for the color b, a contradiction.

We define $\mathcal{B}$ by defining a primitive recursive bijection $\varphi : \omega \to \mathcal{A}$ and then defining $\mathcal{B}$ to be the pullback. To keep φ primitive recursive, we will ensure that $\mathrm{dom}(\varphi)$ is $[0, n)$ at stage $f(n)$.

At stage s, if s is not in the range of f, we define $\varphi_{s+1} = \varphi_s$.

If instead $s = f(n)$, by induction we have $\mathrm{dom}_s = [0, n)$. If $n \in D$, we let $u < f(n)$ be the least element not in the range of φ_s, and we extend φ by $\varphi_{s+1}(n) = u$. If $n \notin D$, we let $u < f(n)$ be the least element not in the range of φ_s such that $c(x, u) = a$ (such a u exists below $f(n)$ by counting), and we extend φ by $\varphi_{s+1}(n) = u$.

Since f has infinite range, it follows that φ is surjective, and thus $\mathcal{B} \cong \mathcal{A}$. On the other hand, by construction we have that $c(\varphi^{-1}(x), n) \neq a \Rightarrow n \in D$. By the sparsity of D, this means there is no primitive recursive bound on the frequency of y with $c(\varphi^{-1}(x), y) = b$ in $\mathcal{B}$, contrary to lemma 3.2. □

Now we can extend the coloring to individual elements, defining $\hat{c}(x) = \lim_y c(x, y)$.

We say that $\hat{c}(\cdot)$ is *almost symmetrically monochromatic* if there is a color a with $a^{-1} = a$ and $\hat{c}(x) = a$ for almost every x.

For a color a, an element $x \in \mathcal{A}$ and a stage s, define $\deg_a(x, s) = \#\{y < s : c(x, y) = a\}$. If we have another structure $\mathcal{B}$, we define $\deg^{\mathcal{B}}_a(x, s)$ similarly, using the atomic diagram of $\mathcal{B}$ to determine colors instead. We define $\deg_a(x) = \lim_s \deg_a(x, s)$, and similarly for $\deg^{\mathcal{B}}_a(x)$.

Lemma 3.4. *If $c(\cdot, \cdot)$ is stable but $\hat{c}(\cdot)$ is not almost symmetrically monochromatic, then*

$$\forall x\, \forall^\infty y\, \forall^\infty s\, \exists a\, [\deg_a(x, s) \neq \deg_a(y, s)].$$

Also, every orbit is finite.

Proof. Fix an x. Let $\hat{c}(x) = a$. There are two cases.

If almost every y is colored a^{-1}, then by assumption $a \neq a^{-1}$, so $\hat{c}(x) \neq a^{-1}$. Then $\deg_{a^{-1}}(x) < \infty$. So for every y with $c(y) = a^{-1}$, for almost every s, $\deg_{a^{-1}}(y, s) > \deg_{a^{-1}}(x) \geq \deg_{a^{-1}}(x, s)$. Also, since almost every element is colored differently from x, x must have finite orbit.

If there are infinitely many elements not colored a^{-1}, then by pigeon hole there is some color b with $b^{-1} \neq a$ such that infinitely many elements are colored b. Since $c(x) \neq b^{-1}$, then $\deg_{b^{-1}}(x) < \infty$. But since there are infinitely many elements colored b, almost every y has $\deg_{b^{-1}}(y) > \deg_{b^{-1}}(x)$. For such a y, for almost every s, $\deg_{b^{-1}}(y, s) > \deg_{b^{-1}}(x) \geq \deg_{b^{-1}}(x, s)$. Since almost every y has $\deg_{b^{-1}}(y) > \deg_{b^{-1}}(x)$, x must have finite orbit. □

Lemma 3.5. *Suppose $c(\cdot, \cdot)$ is stable, but $\hat{c}(\cdot)$ is not almost symmetrically monochromatic. Then $\mathcal{A}$ is not punctually categorical.*

Proof. Once again, we build $\mathcal{B} \cong \mathcal{A}$ by constructing a primitive recursive bijection $\varphi : \omega \to \mathcal{A}$ and defining $\mathcal{B}$ to be the pullback. Again, by a primitive recursive renumbering of stages, we assume that $\mathcal{A}$ is fully defined on $[0, s)$ by stage s. We will have $\mathrm{dom}(\varphi_s)$ either $[0, s-1)$ or $[0, s)$, and $\mathrm{range}(\varphi_s) \subseteq [0, s)$.

We will follow the general framework described in Section 2. Our recovery phase will consist of making φ "catch up", as in the proof of lemma 3.2, so that when we begin the next diagonalization phase at stage $s+1$, we will have $\mathrm{dom}(\varphi_{s+1}) = [0, s+1)$.

At stage 0, we begin the first diagonalization phase.

When we begin the diagonalization phase for the pair (p, q) at some stage s_0, we omit an element from $\mathcal{B}$. That is, we define $\varphi_{s_0+1} = \varphi_{s_0}$, leaving s_0 temporarily out of the range of φ. At subsequent stages s, we extend φ by defining $\varphi_{s+1}(s-1) = s$ until we reach a stage s at which there is a u_0 and an $n_0 < s$ such that $(\varphi \circ p)^{n_0}(s_0) = u_0$ has converged by stage s, and for some color a, $\deg_a(s_0, s) \neq \deg_a(u_0, s)$.

We take a moment to argue why such n_0 and u_0 must eventually be found (under the assumption that p is an isomorphism). Since φ is injective, if p is injective, then $(\varphi \circ p)$ is injective. Since s_0 is not in the range of φ, $(\varphi \circ p)^m(s_0)$ is a sequence with no repetition that grows for as long as we are searching for u_0 and n_0. So by lemma 3.4, we will eventually find such an a, u_0 and n_0.

Suppose we find our desired n_0 and u_0 at stage s. Then we add s_0 to the range of φ. That is, we extend φ by defining $\varphi_{s+1}(s-1) = s_0$. At this and all subsequent stages s, we continue to extend φ by $\varphi_{s+1}(s) = s$ until we reach a stage where $c(s_0, s) \neq c(u_0, s)$. Let $w_0 = s$. At this stage, we omit w_0 from the range of φ, defining $\varphi_{s+1} = \varphi_s$.

We take a moment to argue why such a w_0 must eventually be found (under the assumption that p is an isomorphism). While we are searching for it, we are making φ surjective, and so φ will be an isomorphism. So if p is also an isomorphism, then s_0 and u_0 are in the same orbit and thus have the same color. Since $\sum_c \deg_c(s_0, s) = \sum_c \deg_c(u_0, s) = s$, and we have $\deg_a(s_0, s) \neq \deg_a(u_0, s)$ for the stage s at which we find u_0, by counting there must be at least one other color b with $\deg_b(s_0, s) \neq \deg_b(u_0, s)$. So WLOG, $\hat{c}(s_0) \neq a$, and thus $\hat{c}(u_0) \neq a$. So $\deg_a(s_0)$ and $\deg_a(u_0)$ are finite, and if they're in the same orbit, they must be equal. WLOG, $\deg_a(s_0, s) <$

$\deg_a(u_0, s)$ for the stage s at which we find u_0. So in order to bring these degrees into agreement, there must later appear a w_0 with $c(s_0, w_0) = a$ and $c(u_0, w_0) \neq a$.

Having found w_0 and kept it out of range(φ) temporarily, we will continue for a time defining $\varphi_{s+1}(s-1) = s$. We now argue that while we are doing this, the sequence $(\varphi \circ p)^m(s_0)$ must continue without repetition (under the assumption that p is an isomorphism). Suppose not, and suppose we continue extending φ is in this way forever. Then by injectivity, there must be an m with $(\varphi \circ p)^m(s_0) = s_0$. It follows then that $(\varphi \circ p)^m(u_0) = u_0$. While φ is not currently being built to be an isomorphism, it is being built to be color preserving (as it will omit only a single element), so $\hat{c}(s_0) = \hat{c}(u_0)$. Recall that $c(s_0, w_0) \neq c(u_0, w_0)$, so we have that at least one of the following holds: $c(s_0, w_0) \neq \hat{c}(s_0)$, or $c(u_0, w_0) \neq \hat{c}(u_0)$. WLOG, $c(s_0, w_0) \neq \hat{c}(s_0)$. Let $c(s_0, w_0) = b$.

Note that for all i,

$$\deg_b^{\mathcal{A}}((\varphi \circ p)^i(s_0)) = \deg_b^{\mathcal{B}}(p \circ (\varphi \circ p)^i(s_0)),$$

because p is an isomorphism. Further,

$$\deg_b^{\mathcal{B}}(p \circ (\varphi \circ p)^i(s_0)) = \deg_b^{\mathcal{A}-w_0}((\varphi \circ p)^{i+1}(s_0)),$$

because $\varphi : \mathcal{B} \to \mathcal{A} - w_0$ is an isomorphism. Also,

$$\deg_b^{\mathcal{A}-w_0}((\varphi \circ p)^{i+1}(s_0)) \leq \deg_b^{\mathcal{A}}((\varphi \circ p)^{i+1}(s_0)),$$

because the addition of w_0 can only increase the degree. Further, these values are all finite, because all $(\varphi \circ p)^i(s_0)$ have a color other than b. So by induction, we have that

$$\deg_b^{\mathcal{A}}(s_0) \leq \deg_b^{\mathcal{A}-w_0}((\varphi \circ p)^m(s_0)) = \deg_b^{\mathcal{A}-w_0}(s_0) = \deg_b^{\mathcal{A}}(s_0) - 1.$$

Where the last equality comes from $c(s_0, w_0) = b$. But this is a contradiction.

So while we keep w_0 out of range(φ), the sequence $(\varphi \circ p)^m(s_0)$ must continue without repetition. So eventually we will find a new pair n_1 and u_1 with $(\varphi \circ p)^{n_1}(s_0) = u_1$ and for some color a, $\deg_a(s_0, s) \neq \deg_a(u_1, s)$. When we find this u_1, we permit w_0 to enter the range of φ by defining $\varphi_{s+1}(s-1) = w_0$. We then begin searching for a new w_1 with $c(s_0, w_1) \neq c(u_1, w_1)$, and we omit w_1 from the range of φ. The construction continues in this fashion, constantly obtaining new u_i and w_i, keeping w_i out of the range of φ until such time as we find u_{i+1}, and then repeating.

Thus, if the diagonalization phase never ends, φ is surjective, as no element is ever kept out of the range forever. So φ is an isomorphism. If p is also an isomorphism, $(\varphi \circ p)$ is an automorphism. But we also argued that $(\varphi \circ p)^m(s_0)$ can have no repetition. Thus s_0 has an infinite orbit, while lemma 3.4 says that every orbit of $\mathcal{A}$ is finite, and so $\mathcal{B} \not\cong \mathcal{A}$, and the diagonalization phase must in fact end. □

Lemma 3.6. *Suppose $\hat{c}(\cdot)$ is almost symmetrically monochromatic, but $\mathcal{A}$ is not automorphically trivial. Then $\mathcal{A}$ is not punctually categorical.*

Proof. Let a be the color such that $\hat{c}(x) = a$ for almost every x. Then $a^{-1} = a$ by assumption.

Fix $z_0, \ldots, z_n \in \mathcal{A}$ such that all other elements have color a, and let c_i be the color of z_i. Then for every $\bar{y} \in \mathcal{A} - \bar{z}$, there are infinitely many x with $c(z_i, x) = c_i$ and $c(y, x) = a$ for $y \in \bar{y}$.

We again construct a $\mathcal{B}$ using the general framework from section 2. We will construct a computable isomorphism $\varphi : \mathcal{B} \to \mathcal{A}$. At every stage s of the construction, we will have $\mathcal{B}_s = \operatorname{dom}\varphi_s \sqcup C_s$ for a finite set C_s. $\mathcal{B}_s$ will be defined by pull-back on $\operatorname{dom}\varphi_s$, and for $x \in C_s$ it will be the case that $c(i,x) = c_i$ for $i \leq n$, and $c(y,x) = a$ for $y \in \mathcal{B}_s \setminus [0,n]$.

We begin by defining $\varphi_0(i) = z_i$ for $i \leq n$, and then we enter the first diagonalization phase.

At every stage s of the diagonalization phase, we will let $x = |\mathcal{B}_s|$, the least element not in $\mathcal{B}_s$, and we will define $\mathcal{B}_{s+1} = \mathcal{B}_s \cup \{x\}$, $C_{s+1} = C_s \cup \{x\}$, defining the relations on $\mathcal{B}_{s+1}$ such that $c(i,x) = c_i$ for $i \leq n$ and $c(y,x) = a$ for $y \in \mathcal{B}_{s+1}$. If the diagonalization phase never ends, then $C = \bigcup_s C_s$ is an a-clique, and our structure is automorphically trivial with the finite set $\operatorname{dom}\varphi_{s_0}$. So $\mathcal{B} \not\cong \mathcal{A}$, as required.

During the recovery phase, we continue to expand C_s by a single element at each stage, as we did in the diagonalization phase. This continues until we see an element $w \in \mathcal{A}_s \setminus \operatorname{range}\varphi_s$ with $c(z_i, w) = c_i$ and $c(y,w) = a$ for $y \in \mathcal{A}_s \setminus \bar{z}$. When we find such a w, we let $x = \min C_s$, and we define $\varphi_{s+1}(x) = w$. We define $C_{s+1} = C_s \setminus \{x\}$, and we extend $\mathcal{B}_{s+1}$ and φ_{s+1} such that $[0,s) \subseteq \operatorname{range}\varphi_{s+1}$. We then continue to the next diagonalization phase.

Note that if we have infinitely many recovery phases, then $\lim_s C_s = \emptyset$, and $\operatorname{dom}\varphi = \omega$, so that $\varphi : \mathcal{B} \to \mathcal{A}$ is an isomorphism, as required. □

This completes the proof of Theorem 3.1. □

4. One unary function

This section will frequently employ the sort of argument which appeared in Lemma 3.6. Previous constructions in Section 3 relied on omitting one or several points—a new structure was built that lagged behind the original structure. As we constructed the new structure, we simultaneously constructed a primitive recursive isomorphism from it to the old structure, but this isomorphism was slow to put some points into the range, and so its inverse was not primitive recursive. Now we will build new structures that run ahead of the original structure, as in the proof of Lemma 3.6, adding points before the corresponding points appear in the original structure. We will still build an isomorphism from the new structure to the original structure, but now we will put some points into our new structure significantly before we define the isomorphism on them, and thus the isomorphism will not be primitive recursive.

Theorem 4.1. *Suppose $\mathcal{A}$ is an infinite structure with only one unary functional symbol. Then $\mathcal{A}$ is punctually categorical if, and only if, either:*
1. *$\mathcal{A}$ is almost equal to a disjoint union of loops of identical size,*
2. *$\mathcal{A}$ is almost equal to an infinite star.*

We prove a sequence of lemmas restricting the isomorphism type of such an $\mathcal{A}$.

Lemma 4.2. *Suppose $\mathcal{A}$ is an infinite punctually categorical structure with only one unary function symbol f. Then for every $x \in \mathcal{A}$, there is are $n \neq m$ such that $\mathcal{A} \models f^n(x) = f^m(x)$.*

Proof. Suppose not. Then for some $x \in \mathcal{A}$, the sequence $x, f(x), f^2(x), f^3(x), \dots$ is without repetition. Fix such an x. By applying a bi-primitive recursive permutation to $\mathcal{A}$, we may without loss of generality assume that $x = 0$. By a primitive recursive renumbering of the stages, we may assume that at every stage s, $f^k(a)$ is defined in $\mathcal{A}$ for every $a + k \leq s$.

We will build a punctual structure $\mathcal{B}$ witnessing the failure of punctual categoricity. For clarity, we will use f for the function symbol in $\mathcal{A}$ and g for the function symbol in $\mathcal{B}$. For every primitive recursive function p, we will ensure that $p : \mathcal{B} \to \mathcal{A}$ is not an isomorphism. We will also construct a computable isomorphism $\varphi : \mathcal{B} \to \mathcal{A}$. We will only work at stages of the form $s = t^2$. We will maintain that for every $a \in \operatorname{range} \varphi_s$, there is a b and k with $b + k < s$ and $f^k(b) = a$. Thus $f(a) = f^{k+1}(b)$ is defined by stage s. We will also maintain that $\mathcal{B}_s$ is a proper initial segment of ω, and if $s = t^2$, then $[0, t) \subseteq \mathcal{B}_s$.

We begin by defining $\varphi_1(0) = 0$ and $\mathcal{B}_1 = \{0\}$. At every stage $s > 0$, if s is not of the form $s = t^2$, we define $\varphi_{s+1} = \varphi_s$ and $\mathcal{B}_{s+1} = \mathcal{B}_s$.

Suppose we are at stage $s = t^2$. To defeat the primitive recursive function p, we will define a sequence $y_0, y_1, y_2, \ldots$ with $g(y_k) = y_{k+1}$, but we will keep this sequence out of the domain of φ. As we will see, the domain of φ_s will be precisely $\mathcal{B}_s \setminus \{y_0, y_1, y_2, \ldots\}$.

Let n be least such that $g^n(0)$ is not yet defined in $\mathcal{B}_s$, and let k be least such that y_k is not yet defined. We first check if the following hold:

- $k > 0$ and $n + k < s$;
- $p(0)$ and $p(y_0)$ have converged;
- $f^n(p(0))$ is defined in $\mathcal{A}$ by stage s; and
- $\mathcal{A} \models f^n(p(0)) \neq p(y_0)$.

If this does not hold, let $y_k = |\mathcal{B}_s|$, the least element not in $\mathcal{B}_s$. Let $D = \{f(a) : a \in \operatorname{range} \varphi_s\} \setminus \operatorname{range} \varphi_s$ and $d = |D|$. We let $\mathcal{B}_{s+1} = [0, y_k + d + 1)$, and we extend φ_s to φ_{s+1} via some bijection from $[y_k + 1, y_k + d + 1)$ to D. Thus $\operatorname{dom} \varphi_{s+1} = \mathcal{B}_{s+1} \setminus \{y_0, \ldots, y_k\}$. For $z \in \operatorname{dom} \varphi_s$, we define $g(z) = \varphi_{s+1}^{-1}(f(\varphi_s(z)))$. If $k > 0$, we also define $g(y_{k-1}) = y_k$.

If instead instead the above does hold, we define $g^n(0) = y_0$. By assumption, since $n + k < s$, $f^{n+k}(0)$ is defined in $\mathcal{A}$ at stage s. So we may define θ extending φ_s with $\operatorname{dom} \theta = \operatorname{dom} \varphi_s \cup \{y_0, \ldots, y_{k-1}\}$ with $f^n(0) = \theta(y_0)$ and $f(\theta(y_i)) = y_{i+1}$ for $i < k - 1$. Fix w the least element of $[0, s) \setminus \operatorname{dom} \theta$. Let $D = \{f(a) : a \in \operatorname{range} \varphi_s\} \setminus \operatorname{range} \varphi_s \setminus \{w\}$ and $d = |D|$. We set $\mathcal{B}_{s+1} = [0, |\mathcal{B}_s| + d + 1)$, and we extend θ to φ_{s+1} via some bijection from $[|\mathcal{B}_s| + 1, |\mathcal{B}_s| + d + 1)$ to D, and by $\varphi_{s+1}(|\mathcal{B}_s|) = w$. For $z \in \operatorname{dom} \varphi_s$, we define $g(z) = \varphi_{s+1}^{-1}(f(\varphi_s(z)))$. We have now diagonalized against p, and so we are ready to begin working on the next primitive recursive function.

Since at every stage $s = t^2$, we defined $\mathcal{B}_{s+1}$ to be an initial segment of ω with at least one more element than $\mathcal{B}_s$, we see that $[0, t) \subseteq \mathcal{B}_s$ as promised. Also, since g is defined on all of $\mathcal{B}_s$ by the end of stage s, and $s = t^2$ is a primitive recursive function, $\mathcal{B}$ is a punctual structure.

Suppose we are working against primitive recursive function p. Then at almost every stage $s = t^2$ while we wait for the desired conditions to hold, we will have $k > 0$ and $n + k < s$. This is because at each such stage, k and n grow by precisely 1, while s increases quadratically. Since p is primitive recursive, eventually $p(0)$ and $p(y_0)$ will converge. Then at almost every stage $s = t^2$ while we wait, $f^n(p(0))$ will be defined. Again, this is because s increases quadratically while n increases linearly, and by our assumption about the convergence of f. Of course, there is at most one n with $f^n(p(0)) = p(y_0)$, and so eventually the desired conditions will hold, and we will complete our work for p. Since we will arrange that $g^n(0) = y_0$, while $f^n(p(0)) \neq p(y_0)$, p cannot be an isomorphism.

Since we eventually finish with every primitive recursive function p, by construction we are sure to enumerate all of $\mathcal{A}$ into the range of φ, and thus φ is an isomorphism. So $\mathcal{B} \cong \mathcal{A}$, but there is no primitive recursive isomorphism witnessing this, contrary to assumption. □

So we see that in such an $\mathcal{A}$, every element generates a finite component terminating in a loop.

Lemma 4.3. *Suppose $\mathcal{A}$ is an infinite punctually categorical structure with only one unary function symbol f. If there is an $x \in \mathcal{A}$ for which there are infinitely many $y \in \mathcal{A}$ with $f(y) = x$, then $\mathcal{A}$ is almost equal to an infinite star.*

Proof. Suppose not, and fix an x such that there are infinitely many y with $f(y) = x$. By applying a bi-primitive recursive permutation to $\mathcal{A}$, we may without loss of generality assume that $x = 0$. By a primitive recursive renumbering of the stages, we may assume that at every stage s, $f^k(a)$ is defined in $\mathcal{A}$ for every $a + k \leq s$.

Again we will construct a punctual $\mathcal{B} \cong \mathcal{A}$ witnessing the failure of punctual categoricity. For clarity, we again use g for the unary function in $\mathcal{B}$. For every primitive recursive function p, we will ensure that $p : \mathcal{A} \to \mathcal{B}$ is not an embedding. We will also construct a computable isomorphism $\varphi : \mathcal{B} \to \mathcal{A}$. We will maintain that at every stage s, for every $a \in \operatorname{range} \varphi_s$, there is a b and k with $b + k < s$ and $f^k(b) = a$. Thus $f(a) = f^{k+1}(b)$ is defined by stage s. We will also maintain that $\mathcal{B}_s$ is a proper initial segment of ω with $[0, s) \subseteq \mathcal{B}_s$.

At stage 0, we define $\varphi_1(0) = 0$, $\mathcal{B}_1 = \{0\}$.

At stage s, suppose we are working to diagonalize against p, and s_0 is the stage at which we began considering p. We first check if both of the following hold:

(1) p has proven itself not to be an embedding. That is:
   - There are $a, b < s$ with $a \neq b$, but $p(a) = p(b)$ has converged by stage s; or
   - There are $a, b < s$ with $f(a) = b$, but $p(a), p(b)$ and $g(p(a))$ have converged by stage s with $g(p(a)) \neq p(b)$.

(2) For the least $y \notin \operatorname{dom} \varphi_{s_0}$, $y \in \operatorname{dom} \varphi_s$.

If at least one does not hold, we let $y = |\mathcal{B}_s|$, the least element not in $\mathcal{B}_s$. Let $D = \{f(a) : a \in \operatorname{range} \varphi_s\} \setminus \operatorname{range} \varphi_s$ and $d = |D|$. We let $\mathcal{B}_{s+1} = [0, y+d+1)$, and we extend φ_s to φ_{s+1} via some bijection from $[y+1, y+d+1)$ to D. For $z \in \operatorname{dom} \varphi_s$, we define $g(z) = \varphi_{s+1}^{-1}(f(\varphi_s(z)))$. We also define $g(y) = 0$. If $\mathcal{A} \models f(s-1) = 0$ and $s - 1 \notin D$, we let z be the least element not in $\operatorname{dom} \varphi_s$ (possibly $z = y$), and we also define $\varphi_{s+1}(y) = s - 1$.

If both of the above conditions do hold, we proceed exactly as above. However, if there is an $a < s$ which is not otherwise in the range of φ_{s+1}, we fix the least such a and define $\varphi_{s+1}(y + d + 1) = a$, and redefine $\mathcal{B}_{s+1} = [0, y + d + 2)$. We then proceed to consider the next primitive recursive function.

At every stage, we increase $\mathcal{B}_s$ by at least one element and define g on all elements of $\mathcal{B}_s$, so $\mathcal{B}$ is primitive recursive.

Observe that by construction, if $y \in \mathcal{B}_s \setminus \varphi_s$, then $g(y) = 0$. While we are waiting for condition (2) to hold, we never increase φ except as forced to by f, and so $\operatorname{range} \varphi_s$ is contained in the substructure generated by $\operatorname{range} \varphi_{s_0}$. By Lemma

4.2, this substructure is finite. Since there are infinitely many a with $f(a) = 0$, there will eventually be an $s-1$ with $f(s-1) = 0$ and $s-1 \notin D$. When that happens, we act to satisfy condition (2). So eventually condition (2) will hold for each p we consider.

Suppose that for some p, we never satisfy condition (1). Then, by construction, $\mathcal{B}$ is almost equal to an infinite star. For $\mathcal{B}$ will consist of the finite substructure generated by range φ_{s_0} and infinitely many elements y with $g(y) = 0$. Since $\mathcal{A}$ is not almost equal to an infinite star, there can be no embedding of $\mathcal{A}$ into $\mathcal{B}$. Since p is total, eventually it will prove itself not to be an embedding, and condition (1) will be satisfied. Thus we successfully diagonalize against every primitive recursive function p.

Finally, since we meet condition (2) for every primitive recursive function p we consider, and we proceed to consider every primitive recursive function, $\operatorname{dom}\varphi = \omega$. By our action every time we finish considering a primitive recursive function, $\operatorname{range}\varphi = \omega$. By construction, φ is an isomorphism, and thus $\mathcal{B}$ witnesses the failure of punctual categoricity for $\mathcal{A}$, contrary to hypothesis. □

Lemma 4.4. *Suppose $\mathcal{A}$ is an infinite punctually categorical structure with only one unary function symbol f. If there is some n such that $\mathcal{A}$ contains infinitely many loops of size n, then $\mathcal{A}$ is almost equal to an infinite union of loops of size n.*

Proof. Suppose not, and fix an n such that there are infinitely many loops of size n, but $\mathcal{A}$ is not almost equal to an infinite union of loops of size n. Again we construct a punctual $\mathcal{B}$ witnessing the failure of punctual categoricity. This construction is as the proof of Lemma 4.3, except that instead of adding a new point with $g(y) = 0$ at every stage, we instead add a new loop of size n. □

Lemma 4.5. *Suppose $\mathcal{A}$ is an infinite punctually categorical structure with only one unary function symbol f, and $\mathcal{A}$ is not almost equal to an infinite star or a union of infinitely many loops of a fixed size. Then there are infinitely many elements which are not part of a loop.*

Proof. Suppose not. By a bi-primitive recursive permutation of $\mathcal{A}$, we may assume that there is some m such that $[0, m)$ is the substructure generated by the finitely many points which are not part of a loop. We will again build a punctual $\mathcal{B}$ witnessing the failure of punctual categoricity.

Observe that since there are only finitely many loops of each size (Lemma 4.4), omitting any loop changes the isomorphism type of $\mathcal{A}$. So our plan to diagonalize against a given primitive recursive function $p : \mathcal{A} \to \mathcal{B}$ is to omit a loop disjoint from $[0, m)$ but otherwise copy $\mathcal{A}$ until p demonstrates that it is not an embedding. As soon as we have defeated p, we add the loop we were previously omitting and then proceed to the next primitive recursive function. This is all as in previous arguments, except for the details on how we arrange to omit a loop while maintaining that $\mathcal{B}$ is primitive recursive. So we will explain that and then consider the proof complete.

We define a sequence by $m_0 = 0$, $m_{s+1} = \max\{m_s, f(a) : a \leq m_s\} + 1$. Observe that $s \mapsto m_s$ is primitive recursive. By a primitive recursive renumbering of stages, we may assume that m_s and $f(a)$ for $a \leq m_s$ have converged by stage s.

We will only act at stages of the form $s = t^2$. At such a stage we will define $g(x)$ for every $x \in \mathcal{B}_s$, but we might not add any new points to $\mathcal{B}_s$, resulting in $\mathcal{B}_s = \mathcal{B}_{s+1}$. However, this will not happen at two consecutive such stages s, and so we will always have $[0, t/2) \subseteq \mathcal{B}_s$. As $t \mapsto (2t)^2$ is primitive recursive, this will suffice to make $\mathcal{B}$ primitive recursive.

Suppose we begin considering p at stage $s_0 = t_0^2$. Until we have seen the full substructure generated by range φ_{s_0}, we add no new points except those required to mirror this substructure. At some stage $s_1 = t_1^2$, we will see that the full substructure has revealed itself, and we will have φ_{s_1+1} an isomorphism from $\mathcal{B}_{s_1+1}$ to the substructure generated by range φ_{s_0}, which is contained in $[0, m_{s_1}]$.

Beginning with stage $s_2 = (t_1 + 1)^2$, we build a chain $y_0, y_1, \ldots$ in $\mathcal{B}$, with $g(y_i) = y_{i+1}$. At subsequent stages of the form $s = t^2$, we extend this chain by 1 element. While this is occurring, we do not extend φ. We continue until the first stage $s_3 = (t_3 + 1)^2$ at which $[0, m_s] \setminus \operatorname{range} \varphi_s$ contains a loop.

Fix some $a \in [0, m_{s_2}] \setminus \operatorname{range} \varphi_{s_1+1}$, which is nonempty because it contains $[m_{s_1}, m_{s_2}]$. We observe that at stage s_3, a generates a chain longer than the chain we have so far constructed. For if $s_3 = s_2 = (t_1 + 1)^2$, then we have constructed a chain of length 0, and $\langle a \rangle$ is a chain of length 1. If $s_3 = (t_1 + 2)^2$, then our chain has length 1, and $f(a) \neq a$ (as otherwise it is a loop, contrary to choice of s_3), so $\langle a, f(a) \rangle$ is a chain of length 2. If the loop occurred at a stage $s_3 = (t_3 + 1)^2$ for $t_3 \geq t_1 + 2$, then since there was no loop at stage t_3^2, and $t_3^2 - s_2^2 > t_3$, it must be that $\langle a, f(a), f^2(a), \ldots, f^t(a) \rangle$ contains no repeats, and so is a chain of length $t_3 + 1$, while our chain has length at most t_3.

If there is a loop in $[0, m_{s_3}] \setminus \operatorname{range} \varphi_{s_3}$ which is disjoint from this chain generated by a, then this is the loop we will omit. We define φ_{s_3+1} to send the chain we have constructed to the chain generated by a. At subsequent stages $s > s_3$, $[0, m_s]$ will contain new elements, and these new elements will necessarily not generate the omitted loop (as they are part of disjoint loops, by assumption), so we can copy those elements while continuing to omit the loop.

If the chain generated by a is part of the only loop in $[0, m_{s_3}] \setminus \operatorname{range} \varphi_{s_3}$, then this is the situation in which we define $\mathcal{B}_{s_3+1} = \mathcal{B}_{s_3}$. In particular, we do not extend our chain at this stage. Now consider $b = m_{s_3} + 1$, and let $s_4 = (t_3 + 2)^2$, the next stage after s_3 at which we act. Since $s_4 - (s_3 + 1) > t_3$, we see that either b generates a loop in $[0, m_{s_4}]$ or it generates a chain longer than that which we have constructed. In the former case, we can now map our chain to the chain generated by a (adding at least one element, since the latter chain is strictly longer) and decide that the loop generated by b is the one we omit; in the latter case, we can map our chain to the chain generated by b (again adding at least one element) and decided that the loop generated by a is the one we omit.

In this way we have arranged to omit a loop while only occasionally having a stage at which we do not add a new element to $\mathcal{B}$. □

Lemma 4.6. *Suppose $\mathcal{A}$ is an infinite punctually categorical structure with only one unary function symbol f, and $\mathcal{A}$ is not almost equal to an infinite star or a union of infinitely many loops of a fixed size. Then for every proper substructure $\mathcal{B} \subset \mathcal{A}$, $\mathcal{B} \ncong \mathcal{A}$.*

Proof. Fix $\mathcal{B}$ a proper substructure. For each $x \in A$, let $m(x)$ and $n(x)$ be the least $0 \leq m < n$ such that $\mathcal{A} \models f^m(x) = f^{m+n}(x)$. Note that $n(x)$ is the size of the loop generated by x, m is the minimum distance from x to the loop it generates, and, if $x \in \mathcal{B}$, then $m(x)$ and $n(x)$ are also the least $0 \leq m < n$ such that $\mathcal{B} \models f^m(x) = f^{m+n}(x)$.

By Lemmas 4.3 and 4.4, the sets

$$D_{n,m} = \{x \in \mathcal{A} : m(x) = m \ \& \ n(x) = n\}$$

are all finite. Fix a $z \in \mathcal{A} \setminus \mathcal{B}$. Then for $n = n(z)$ and $m = m(z)$, $D_{n,m} \cap \mathcal{B}$ is smaller than $D_{n,m}$, and so $\mathcal{B} \ncong \mathcal{A}$. □

Lemma 4.7. *Suppose $\mathcal{A}$ is an infinite punctually categorical structure with only one unary function symbol f, and $\mathcal{A}$ is not almost equal to an infinite star or a union of infinitely many loops of a fixed size. Then for almost every $x \in \mathcal{A}$, there is a $y \in \mathcal{A}$ with $f(y) = x$.*

Proof. We construct punctual $\mathcal{B} \cong \mathcal{A}$ while attempting to diagonalize against bi-primitive recursive isomorphism. We again build a computable isomorphism $\varphi : \mathcal{B} \to \mathcal{A}$ and let $\mathcal{B}$ be defined via pullback. Our strategy for defeating a pair (p, q) of primitive recursive functions is to only copy the even elements of $\mathcal{A}$. That is, assuming we begin working to defeat (p, q) at stage s_0, we let $\mathcal{C} \subseteq \mathcal{A}$ be the substructure generated by $\operatorname{range} \varphi_{s_0} \cup 2\mathbb{N}$, and we extend φ such that $\operatorname{range} \varphi = \mathcal{C}$ until we have defeated (p, q). Since $s \mapsto 2s$ is primitive recursive, this keeps $\mathcal{B}$ punctual. Once we have defeated (p, q) at some stage s_1, we define φ_{s_1+1} by extending φ_{s_1} such that the range contains all the odd elements of $\mathcal{A}$ below s_1 and then begin to work against the next pair.

Since $\mathcal{A}$ is punctually categorical, consider the first pair (p, q) which we are not able to defeat by the above strategy, and let s_0 be the stage we begin working for this pair. Then we forever copy $\mathcal{C}$, so $\mathcal{B} \cong \mathcal{C}$. So it must be that $\mathcal{A} \cong \mathcal{C}$. By Lemma 4.6, $\mathcal{A} = \mathcal{C}$, and in particular every odd element of $\mathcal{A}$ is an element of $\mathcal{C}$. So for every odd element $x \in \mathcal{A}$ outside of $\operatorname{range} \varphi$, there must be a y with $f(y) = x$.

By repeating the same construction with the roles of even and odd interchanged, the result follows. □

Proof of Theorem 4.1. That an $\mathcal{A}$ of either of the two isomorphism types is punctually categorical is a simple back-and-forth argument.

For the converse, suppose $\mathcal{A}$ is punctually categorical but not of one of the listed isomorphism types. By Lemma 4.7, there are only finitely many elements $x \in \mathcal{A}$ for which there does not exist a $y \in \mathcal{A}$ with $f(y) = x$. Let $\mathcal{C}$ be the substructure generated by these finitely many elements. By Lemma 4.2, $\mathcal{C}$ is finite. By Lemma 4.5, there are infinitely many elements which are not part of a loop, so fix $z \in \mathcal{A} \setminus \mathcal{C}$ which is not part of a loop. Then z is the head of an infinite backwards-chain – there are $z_0 = z, z_1, z_2, z_3, \ldots$ with $f(z_{i+1}) = z_i$ for all i. Further, by $z \notin \mathcal{C}$, for any y with $f^k(y) = z$ for some k, y is the head of its own infinite backwards-chain. We again assume that $z = 0$.

We will build punctual $\mathcal{B} \cong \mathcal{A}$ contradicting punctual categoricity. We again build a computable isomorphism $\varphi : \mathcal{B} \to \mathcal{A}$, beginning with $\varphi(0) = 0$. Suppose we begin working to defeat the pair (p, q) at stage s_0. Our strategy is to only put into $\operatorname{range} \varphi_s$ those elements generated by $\operatorname{range} \varphi_{s_0}$. Meanwhile, we construct a forward chain $\langle x_0, x_1, x_2, \ldots \rangle$ with $g(x_i) = x_{i+1}$ for all i, and $\mathcal{B}_s$ will be the disjoint union of $\operatorname{dom} \varphi_s$ and the chain. In this fashion we are generating a structure which is almost equal to an infinite chain, and so by Lemma 4.2, we must eventually defeat the pair (p, q).

We continue in this fashion until some stage s_1 at which we have both defeated (p, q), and the range of φ_{s_1} is closed under f. Let x_k be the last element of the forward-chain which has been built by stage s_1. We choose a $y \in \operatorname{dom} \varphi_{s_1}$ with $g^k(y) = 0$ but such that there is no $w \in \operatorname{dom} \varphi_{s_1}$ with $g(w) = y$ (such a y exists as φ_{s_1} is finite and contains 0, and we allow $k = 0$), and we define $g(x_k) = y$ (and thus we stop building this chain). We then begin building a new forward chain while we wait for a stage at which we can extend φ_{s_1} to the previous forward chain. By the above discussion, there will be such a stage. Once we can, we extend φ_{s_1} to include the old chain and such that the range also contains all elements of $\mathcal{A}$ below s_1, and then we begin working to defeat the next pair, using the already in progress forward chain. □

5. Several unary symbols

Theorem 5.1. *Every punctually categorical structure with only unary function symbol and relation symbols is* PA$(0')$*-categorical.*

Note that every finitely generated structure is computably categorical, and thus PA$(0')$-categorical, so we may consider only structures which are not finitely generated. Note also that, as we are considering structures with only unary function symbols, the substructure generated by a set F is the union of the substructures generated by individual elements $x \in F$. In particular, if a finite F generates an infinite substructure, then some element of F generates an infinite substructure.

Lemma 5.2. *Suppose $\mathcal{A}$ is punctually categorical with only unary function and relation symbols, and $\mathcal{A}$ is not finitely generated. Then every finite subset of $\mathcal{A}$ generates a finite substructure.*

Proof. Suppose not, and fix an element x generating an infinite substructure $\mathcal{C}$. We construct a punctual $\mathcal{B} \cong \mathcal{A}$ using the general framework from section 2. We also build a computable isomorphism $\varphi : \mathcal{B} \to \mathcal{A}$, and we begin with $\varphi_0(0) = x$.

If we begin a diagonalization phase at stage s_0, our strategy for the diagonalization is to only add the elements required by range φ_{s_0}. That is, if $\mathcal{C}_0$ is the substructure generated by range φ_{s_0}, we keep $\mathcal{B}_s = \operatorname{dom} \varphi_s$ and range $\varphi_s \subset \mathcal{C}_0$. As $\mathcal{C}_0$ contains x and is thus infinite, this keeps $\mathcal{B}$ punctual. If the diagonalization phase never ends, then $\mathcal{B} \cong \mathcal{C}_0$, and $\mathcal{C}_0 \not\cong \mathcal{A}$, as $\mathcal{C}_0$ is finite generated, so $\mathcal{B} \not\cong \mathcal{A}$, as required.

The recovery phase consists of a single stage at which we extend φ_{s+1} to contain all elements of $\mathcal{A}$ below s, and define $\mathcal{B}_{s+1} = \operatorname{dom} \varphi_{s+1}$ by pull-back. Thus if we have infinitely many recovery stages, range $\varphi = \mathcal{A}$, and so $\mathcal{B} \cong \mathcal{A}$. □

For a structure $\mathcal{A}$ with only unary function and relation symbols, and $x \in \mathcal{A}$, we will adopt the notation $\langle x \rangle_{\mathcal{A}}$ for the substructure of $\mathcal{A}$ generated by x.

Lemma 5.3. *Suppose $\mathcal{A}$ is punctually categorical with only unary function symbols and relations, and there is a finite isomorphism type which occurs as a substructure of $\mathcal{A}$ infinitely often. Then $\mathcal{A}$ is computably categorical.*

Proof. Fix an $x \in \mathcal{A}$ generating a finite substructure such that the isomorphism type of $\langle x \rangle_{\mathcal{A}}$ occurs infinitely many times in $\mathcal{A}$. Assume that $|\langle x \rangle_{\mathcal{A}}|$ is minimal with this property: there is no y such that the isomorphism type of $\langle y \rangle_{\mathcal{A}}$ occurs infinitely many times in $\mathcal{A}$ and $|\langle y \rangle_{\mathcal{A}}| < |\langle x \rangle_{\mathcal{A}}|$. So by pigeon hole, there are only finitely many y such that $\langle y \rangle_{\mathcal{A}}$ is isomorphic to a proper substructure of $\langle x \rangle_{\mathcal{A}}$.

We next form a sort of Δ-system of copies of $\langle x\rangle_{\mathcal{A}}$. First, consider the set

$$F = \{y \in \mathcal{A} : \exists x_1, x_2\, [\langle x_1\rangle_{\mathcal{A}} \cong \langle x_2\rangle_{\mathcal{A}} \cong \langle x\rangle_{\mathcal{A}} \;\&\; x_2 \notin \langle x_1\rangle_{\mathcal{A}} \;\&\; y \in \langle x_1\rangle_{\mathcal{A}} \cap \langle x_2\rangle_{\mathcal{A}}]\}.$$

Note that for $\langle x_1\rangle_{\mathcal{A}} \cong \langle x_2\rangle_{\mathcal{A}} \cong \langle x\rangle_{\mathcal{A}}$, $x_2 \notin \langle x_1\rangle_{\mathcal{A}}$ is equivalent to stating that $\langle x_1\rangle_{\mathcal{A}}$ and $\langle x_2\rangle_{\mathcal{A}}$ are not identical as sets. For such a y, since x_1 and x_2 both generate y, but neither generates the other, y cannot generate either. So $\langle y\rangle_{\mathcal{A}}$ is a proper substructure of $\langle x_1\rangle_{\mathcal{A}}$, and is thus isomorphic to a proper substructure of $\langle x\rangle_{\mathcal{A}}$. It follows that F is finite.

By pigeon hole, we may fix a $D \subseteq F$ such that there are infinitely many x' with $\langle x'\rangle_{\mathcal{A}} \cong \langle x\rangle_{\mathcal{A}}$ and $\langle x'\rangle_{\mathcal{A}} \cap F = D$. Without loss of generality, we may assume that $\langle x\rangle_{\mathcal{A}} \cap F = D$. Now consider the collection of all z such that $x \mapsto z$ induces an isomorphism $\langle x\rangle_{\mathcal{A}} \to \langle z\rangle_{\mathcal{A}}$, and such that $\langle z\rangle_{\mathcal{A}} \cap F = D$. Note that this is a stronger condition than merely that $\langle x\rangle_{\mathcal{A}} \cong \langle z\rangle_{\mathcal{A}}$, but it is still an infinite collection.

For each such z, let $D_z \subset \langle x\rangle_{\mathcal{A}}$ be the preimage of D under the map induced by $x \mapsto z$. By pigeon hole, there is a set D_0 such that $D_0 = D_z$ for infinitely many z in our collection. Pass to the subcollection of all z with $D_0 = D_z$, and without loss of generality assume that x is in this subcollection. Thus $D_0 = D$, and so the maps $x \mapsto z$ fix D as a set.

For each z in our collection, let σ_z be the permutation of D induced by $x \mapsto z$. By pigeon hole again, there is a permutation σ of D such that $\sigma = \sigma_z$ for infinitely many z in our collection. Pass to the subcollection of z with $\sigma = \sigma_z$, and without loss of generality assume that x is in this subcollection. Then $\sigma = \mathrm{id}_D$, and so the maps $x \mapsto z$ fix D pointwise. Call our final collection C.

Claim 5.4. *$\mathcal{A}$ is almost isomorphic to $\bigcup_{z\in C} \langle z\rangle_{\mathcal{A}}$.*

Proof. This is as the proof of Lemma 4.3. Suppose not. We construct a punctual structure $\mathcal{B} \cong \mathcal{A}$ witnessing the failure of punctual categoricity. We begin with a single copy of $\langle x\rangle_{\mathcal{A}}$. During the diagonalization phase, we place more $\langle z\rangle_{\mathcal{A}}$ such that $x \mapsto z$ is an isomorphism fixing D pointwise. If a diagonalization phase never ends, then we produce $\mathcal{B} \cong \bigcup_{z\in C} \langle z\rangle_{\mathcal{A}} \ncong \mathcal{A}$, as required. During a recovery phase we add in the missing elements and extend the isomorphism to include a new $\langle z\rangle_{\mathcal{A}}$, so if there are infinitely many recovery phases, $\mathcal{B} \cong \mathcal{A}$, as required. □

It now follows that $\mathcal{A}$ is computably categorical by a simple back-and-forth construction, completing the proof of Lemma 5.3. □

Proof of Theorem 5.1. Suppose $\mathcal{A}$ has only unary function symbols and relations and is punctually categorical. By our previous results and discussion, we need only consider the case where every element of $\mathcal{A}$ generates a finite substructure, and no finite isomorphism type occurs as a substructure of $\mathcal{A}$ infinitely often.

Suppose $\mathcal{B} \cong \mathcal{A}$ is computable. Of course, given $x \in \mathcal{A}$ and $y \in \mathcal{B}$, for $x \mapsto y$ to be extendible to an isomorphism, a necessary condition is that $\langle x\rangle_{\mathcal{A}} \cong \langle y\rangle_{\mathcal{B}}$. By assumption, there are only finitely many $y \in \mathcal{B}$ with $\langle y\rangle_{\mathcal{B}} \cong \langle x\rangle_{\mathcal{A}}$. So consider the tree of pairs

$$\begin{aligned} T = \{(\sigma,\tau) \in \omega^{<\omega} : |\sigma| = |\tau| & \\ & \;\&\; \sigma : \mathcal{A} \to \mathcal{B} \text{ preserves all atomic formula} \\ & \;\&\; \forall x < |\sigma|\, [\langle x\rangle_{\mathcal{A}} \cong \langle \sigma(x)\rangle_{\mathcal{B}}] \\ & \;\&\; \forall y < |\tau|\, [\tau(y) < |\sigma|] \to \sigma(\tau(y)) = y\}. \end{aligned}$$

Then T is a computable, finitely branching tree such that $[T]$ is the space of isomorphisms from $\mathcal{A}$ to $\mathcal{B}$. We do not know that there is a computable bound on the branching factor of T, but there is necessarily a $0'$-computable bound, and thus every degree in $\mathrm{PA}(0')$ computes an element of $[T]$. □

6. Proof of Theorem 1.6

6.1. **A plan of the proof.** We will construct a punctually categorical $\mathcal{A}$ and a computable $\mathcal{B} \cong \mathcal{A}$ such that every isomorphism between $\mathcal{A}$ and $\mathcal{B}$ computes $0''$.

Fix a Π^0_2-complete predicate P. It is straightforward to construct a primitive recursive function h satisfying the following:

- For all i and s, $1 \leq h(i,s) \leq h(i,s+1)$;
- There are infinitely many i with $\lim_s h(i,s) = \infty$;
- For each n with $0 < n < \omega$, there is exactly one i with $\lim_s h(i,s) = n$; and
- $\lim_s h(2i,s) = \infty$ iff $i \in P$.

We define $h(i) = \lim_s h(i,s)$.

We will have a sequence $(r_i)_{i\in\omega}$ of points in $\mathcal{A}$, and a sequence of isomorphism types $(C_j)_{j\in\omega}$. Each r_i will have an attached copy of C_j, for every j with $j-1 < h(i)$. So the (unique) i with $h(i) = 1$ will have attached copies of C_0 and C_1, the (unique) i with $h(i) = 2$ will have attached copies of C_0, C_1 and C_2, etc, and each of the (infinitely many) i with $h(i) = \infty$ will have an attached copy of each of the C_j. Each i have its own copy of the relevant C_j.

The map $r_i \mapsto i$ will be primitive recursive, although its inverse will not be, as we will be very slow in placing the points r_i. Although we defer the full description of the C_j for the moment, and they will be determined dynamically over the course of the construction, we mention now that it will be unambiguous when a point belongs to a copy of C_j; we will place that point with the intention of it being part of C_j (and with the intention of which r_i that C_j is attached to), and we will never change our mind.

Supposing we have built $\mathcal{A}$ according to the above plan, we now describe how to build computable $\mathcal{B}$. We construct $\mathcal{B}$ as $\mathcal{A}$, again with a computable sequence $(r_i)_{i\in\omega}$, except that in $\mathcal{B}$, r_{2i} has attached copies of C_j for $j \leq i+1$, and r_{2i+1} has attached copies of C_j for all $j < \omega$. Constructing $\mathcal{B}$ computably is straightforward: to determine how C_j should look, we simply wait until we find some i and s with $h(i,s) \geq j$, and then we look at the copy of C_j attached to r_i in $\mathcal{A}$.

Observe that $\mathcal{B}$ is isomorphic to $\mathcal{A}$, but for any isomorphism $\iota : \mathcal{A} \to \mathcal{B}$, if $\iota(r^{\mathcal{A}}_{2i}) = r^{\mathcal{B}}_k$, then k is odd iff $i \in P$. Thus any isomorphism between $\mathcal{B}$ and $\mathcal{A}$ will compute $0''$.

It remains to construct $\mathcal{A}$ according to this general plan while making $\mathcal{A}$ punctually categorical. We must have a way of meeting the requirements:

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

where $(P_e)_{e\in\omega}$ is the natural uniformly computable listing of all punctual structures. Clearly, the list itself is not primitive recursive, for otherwise we would be able to produce a punctual structure which is not in the list. The reader should think of P_e as of being "increasingly slow" in e. However, we will argue that for each fixed e there is a primitive recursive time-function, i.e., a function that bounds the speed of approximation of $P_e = \bigcup_s P_{e,s}$ within the overall uniform primitive recursive approximation $(P_{e,s})_{e,s\in\omega}$. We take this property for granted throughout the proof; see the Appendix of [BDKM19] for a formal clarification.

6.2. **The first part of our language.** We will introduce our construction in several parts. These parts will require various symbols in the language (mostly unary and binary function symbols), and so we will describe our language in pieces, introducing in each part the symbols necessary for that part of the construction.

We begin with a constant d. The purpose of this constant is to be a "dump" for unimportant function values. That is, whenever we fail to define a function on a given tuple, our intention is that the function takes value d for that tuple.

6.3. **Chains.** We will use the method of chains first introduced in [KMN17]. This is a method for building components consisting of long chains with attached loops while preserving punctual categoricity. This technique makes use of three unary function symbols, which we will not bother to name. Although we refer the reader to the earlier paper for a full description, we summarize the salient points of the technique here:

(1) A chain is generated by any single element from the chain.
(2) There is a unary function mapping every point of the chain to its first element, so in a punctual structure, the first element of a chain can be quickly obtained from any element.
(3) There is a distinguished last element of each chain, which is immediately recognizable. Once this last element is placed, the chain is complete, and no more elements can be added to it. The decision to finish a chain is not reversible.
(4) In order to employ this technique, once we begin a chain, we must build that chain to the exclusion of all else. We may not place any points in the structure outside of the chain until have we placed the last element of the chain.
(5) In order to employ this technique, each chain must be very large relative to the stage at which it was begun. That is, the function $s \mapsto$ (the number of elements in the next chain begun after stage s) will dominate any primitive recursive function. However, there is no upper limit on how large we may choose to make a chain; if we wish to waste time, we may continue constructing a given chain indefinitely.
(6) When we begin a chain, we may name any finite number of punctual structures P_e. For each of these P_e, the construction will either isomorphically map our chain to a copy of itself in P_e or prove that P_e is not isomorphic to our structure. Further, the isomorphism type of the chain (as determined by the sizes of the attached loops) will be distinct from that of any chain which occurred in P_e prior to us beginning our chain, up to a primitive recursive tolerance.[3]

We will use chains in three contexts. We will use them as *labels*, we will use them to *delay*, and we will also string together infinitely many of them to construct each C_i.

[3]That is, there is a primitive recursive function p depending on e such that any chain which includes even a single element of P_e prior to stage s will be distinct from any chain we construct beginning after stage $p(s)$.

6.4. **Placing the r_i.** Our intention is to place an infinite sequence $(r_i)_{i\in\omega}$ of points which are all potentially in the same orbit. We will be placing these slowly, but our opponent (constructing a punctual copy of $\mathcal{A}$) might place their own points quickly, simply by repeatedly copying r_0. If this were to happen, we would be unable to construct a primitive recursive isomorphism from our opponent's copy to $\mathcal{A}$, as we would be unable to quickly map these additional r_i. So we must have a strategy to prevent our opponent from doing this.

6.4.1. *Idea.* We will give each nonempty finite set of r_i a label, which will be a chain created for this purpose. As we must not interfere with the possibility of the r_i being in the same orbit, we will give the same label to every set of the same size. Here we do not mean separate labels of the same isomorphism type; for a given k, each set of k distinct r_i will point to the same element, and that element will be the first element of our chain. Creating the label for sets of size $k+1$ will be the first thing we do after placing the element r_k, and we will map the set $\{r_0, \dots, r_k\}$ to it.

If our opponent shows us $k+1$ distinct r_i in P_e much before we have placed r_k, then they will have shown us at least some of a label attached to a set of size $k+1$. When we eventually place r_k and build the label for sets of size $k+1$, as we will be using the chain method to construct this label, it will be of a different isomorphism type to the label occurring in P_e, and so we will have ensured that $\mathcal{A} \not\cong P_e$.

Of course, we are only permitted a finite signature, and each is of a fixed arity, so we cannot directly map arbitrary finite sets to labels. We will instead use a binary function to build sequences via pairing.

6.4.2. *The language needed.* The sublanguage of $\mathcal{A}$ we will need for this strategy consists of:

- A binary function symbol f.
- Two unary function symbols b_1, b_2.
- A unary function symbol l.

6.4.3. *The strategy.* We will have elements x_σ for every nonempty, nonrepeating sequence σ of the r_i. For $\sigma = \langle r_i \rangle$, $x_\sigma = r_i$. Otherwise, each x_σ will be distinct from all other elements mentioned in the construction.

For any such sequence σ, and any r_i not occurring in σ, $f(x_\sigma, r_i) = x_{\sigma^\frown r_i}$.

For any such sequence $\sigma = \tau^\frown r_i$, $b_1(x_\sigma) = x_\tau$ and $b_2(x_\sigma) = r_i$. If τ is empty, $b_1(x_\sigma) = d$.

For any such sequence σ, $l(x_\sigma)$ is the first element of the label for sets of size $|\sigma|$. If σ and τ are both such sequences with $|\sigma| = |\tau|$, then $l(x_\sigma) = l(x_\tau)$.

Observe that the x_σ are distinguished as the unique elements y with $b_2(y) \neq d$, while the r_i are distinguished as the unique elements y with $b_2(y) = y$ and $y \neq d$.

When we place the point r_k, we will immediately place x_σ for every nonrepeating sequence σ from $\{r_0, \dots, r_k\}$ that includes r_k (x_σ for nonempty sequences omitting r_k having already been placed), and we will define f, b_1 and b_2 appropriately. Since we are only considering nonrepeating sequences, there are only a finite number of such sequences, and indeed the number is given by a primitive recursive function in k. As we will be placing r_k at some stage $s \geq k$, this is only a small number of points being placed at this stage.

For every σ containing r_k with $|\sigma| < k+1$, the label for sets of size $|\sigma|$ will have already been created, so we define $l(x_\sigma)$ appropriately. We then begin creating the label for sets of size $k+1$ and define $l(x_\sigma)$ to map to the first element of this label, for $|\sigma| = k+1$.

6.4.4. *Why it works.* Suppose that a P_e we are watching shows us some element y with $b_2(y) \neq d$, and we have so far placed $r_0, \ldots, r_k$. As we will later argue, we will be able to isomorphically map the r_i of $\mathcal{A}$ into P_e punctually, and from that we will be able to map all the x_σ generated by $r_0, \ldots, r_k$ (using that the number of such σ, and the number of steps in their generation, is small relative to the current stage).

If y is not in the image of the map we have so far constructed, then we avoid placing r_{k+1} for the time being while we perform certain calculations in P_e. However, we must continue adding elements of $\mathcal{A}$ in order to make $\mathcal{A}$ punctual. If we are in the midst of constructing some chain, we can simply continue growing that chain indefinitely while we wait. If we have just finished a chain, we can instead begin a new chain for no purpose other than to keep the construction occupied, and again continue that indefinitely while we wait. This second situation is what we meant earlier when we said that chains would be used for delaying.

In the meantime, we consider $b_2(y), b_2(b_1(y)), b_2(b_1^2(y)), \ldots, b_2(b_1^{k+1}(y))$ in P_e. Either this list contains an element z with $P_e \models b_2(z) = z \wedge z \neq d$ and with z not in the image of our map (that is, some new point of r_i-type), or P_e will have proven itself not to be isomorphic to $\mathcal{A}$. In the former case, we can use f in P_e to generate an x_σ with $|\sigma| = k+2$, and then we can calculate $l(x_\sigma)$. Once we have found this element, we no longer avoid placing r_{k+1} (so we finish the current chain whenever the chain technique permits us to, and then we proceed with the construction). When r_{k+1} is eventually placed, the label for sets of size $k+2$ will only then be constructed, and it will be different from the label which $l(x_\sigma)$ is a part of in P_e, by 6.3(6), and thus we will have ensured that $\mathcal{A} \ncong P_e$.

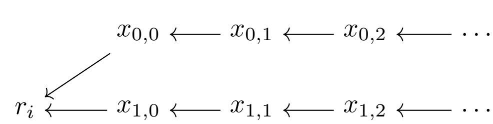
So if $\mathcal{A} \cong P_e$, then no element of type x_σ can occur in P_e before we place it in $\mathcal{A}$, as desired.

6.5. **Building the C_j.** Each C_j will have the form

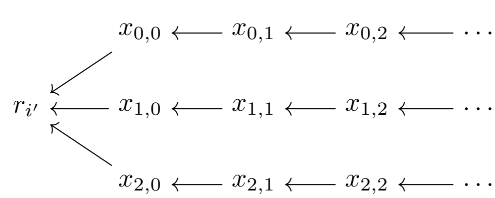
$$x_{j,0} \leftarrow x_{j,1} \leftarrow x_{j,2} \leftarrow \ldots$$

where each $x_{j,k}$ is a chain, and the arrow indicates an unnamed unary function mapping the terminal element of chain $x_{j,k+1}$ to the initial element of chain $x_{j,k}$. This same function will map the terminal element of $x_{j,0}$ to r_i, for whichever r_i this copy of C_j is attached to.

Suppose $h(i) = 1$. Then the picture for r_i will be



Similarly, if $h(i') = 2$, then the picture for $r_{i'}$ will be



In general, if $j-1<\min\{h(i),h(i')\}$, then in the final structure, the copy of C_j attached to r_i will be identical (but disjoint) to the copy attached to $r_{i'}$. At every stage of the construction, however, the two copies will look distinct. At some stage s after we have placed r_i and $r_{i'}$, we will have the following picture (other $C_{j'}$ have been omitted for clarity):

$$r_i \longleftarrow x_{j,0} \longleftarrow x_{j,1} \longleftarrow \cdots \longleftarrow x_{j,k}$$

$$r_{i'} \longleftarrow x_{j,0} \longleftarrow x_{j,1} \longleftarrow \cdots \longleftarrow x_{j,k'}$$

Here $k \neq k'$, so the two copies of C_j will be of different lengths. Throughout the construction we will extend the lengths of one or the other, and the lengths will "leapfrog" past each other.

6.5.1. *Placing* $x_{j,n}$. When we place the first copy of $x_{j,n}$, inside the C_j attached to some r_i, we construct it via the chain technique, and this determines its isomorphism type. We will later place other copies of the same $x_{j,n}$ within other copies of C_j. When we do, the size of $x_{j,n}$ will be small relative to the current stage, and so we will not be able to create it via the chain technique. Also, the chain technique produces a dynamically determined isomorphism type, and we have an intended isomorphism type in mind. So we need a different strategy to place these copies of $x_{j,n}$ while maintaining punctual categoricity.

Our approach is to instead construct $x_{j,n+1}$ via the chain technique. As we place the terminal element of this first copy of $x_{j,n+1}$, we will simultaneously place the entirety of the new copy of $x_{j,n}$ (recalling that this is a small number of points relative to the current stage) and have the terminal point of $x_{j,n+1}$ point to the initial point of this new $x_{j,n}$.

For any P_e we are watching, our chain technique will have isomorphically mapped $x_{j,n+1}$ into P_e, and in particular will have mapped the terminal element. We can then follow the function from the terminal element in P_e to P_e's copy of the new $x_{j,n}$. Since this new copy is small, we can quickly generate the entire $x_{j,n}$ in P_e (or see that $P_e \not\cong \mathcal{A}$) and then map the entirety of our new $x_{j,n}$ to P_e's copy, in an appropriate fashion.

We illustrate the steps of this process, with $n=2$. We place the original copy of $x_{j,n}$ in the C_j attached to r_i, via the chain technique. During the intermediate steps of the chain technique, $x_{j,2}$ is partially constructed and does not yet point to $x_{j,1}$, as it is the terminal element of a chain which points.

$$r_i \longleftarrow x_{j,0} \longleftarrow x_{j,1}$$

$$\Big\Downarrow$$

$$r_i \longleftarrow x_{j,0} \longleftarrow x_{j,1} \qquad x_{j,2}$$

$$\Big\Downarrow$$

$$r_i \longleftarrow x_{j,0} \longleftarrow x_{j,1} \longleftarrow x_{j,2}$$

Then, when we wish to attach a copy of $x_{j,2}$ to the C_j of some $r_{i'}$, we build $x_{j,3}$. During the intermediate steps of the chain construction, $x_{j,3}$ is partially constructed and does not yet point to $x_{j,2}$, and indeed the $x_{j,2}$ is not yet built. We place the entire $x_{j,2}$ in a single step at the end:

$$r_i' \longleftarrow x_{j,0} \longleftarrow x_{j,1}$$

$$\Downarrow$$

$$r_i' \longleftarrow x_{j,0} \longleftarrow x_{j,1} \qquad\qquad\qquad x_{j,3}$$

$$\Downarrow$$

$$r_i' \longleftarrow x_{j,0} \longleftarrow x_{j,1} \longleftarrow x_{j,2} \longleftarrow x_{j,3}$$

6.5.2. *Full extension.* The above was a slight simplification, because when we wish to place a subsequent copy of $x_{j,n}$ for some r_i, it may be that a copy of $x_{j,n+1}$ already exists, and so we cannot build it now via the chain method. Instead, we will have to choose the least k such that $x_{j,k}$ does not yet exist anywhere in the structure, and build $x_{j,k}$. When it is completed, we will simultaneously place the entire copy of

$$x_{j,n} \leftarrow x_{j,n+1} \leftarrow \cdots \leftarrow x_{j,k-1},$$

and have the terminal element of the new $x_{j,k}$ point to the first element of $x_{j,k-1}$. We can extend the construction of $x_{j,k}$ so as to bide our time until this is a small number of points to place all at once. We refer to this process as *extending C_j for r_i*.

6.5.3. *Placing an r_k by extending C_0.* For C_0 only, we will allow ourselves to extend C_0 for r_k before we have placed r_k. In fact, this will be the method by which r_k is placed in the structure. This means that we choose the least k such that $x_{0,k}$ does not yet exist anywhere in the structure, and we build $x_{0,k}$. When it is completed, we will simultaneously place the entire copy of

$$r_k \leftarrow x_{0,0} \leftarrow x_{0,1} \leftarrow \cdots \leftarrow x_{0,k-1},$$

and have the terminal element of the new $x_{0,k}$ point to the first element of $x_{0,k-1}$. If $k = 0$, we instead have the terminal element of the new $x_{0,0}$ point to r_k.

6.6. **Ensuring new chains attach to the correct r_i.** Suppose we wish to extend C_j for some r_i, as described above, in the process using the chain technique to construct some $x_{j,n}$. If P_e is one of the punctual structures we are watching, then when we have completed construction of $x_{j,n}$, the chain machinery will have mapped $x_{j,n}$ punctually isomorphically into P_e. From the terminal element of the chain in P_e we can obtain the initial element of a copy of $x_{j,n-1}$, from which we can quickly generate the entire copy of $x_{j,n-1}$ in P_e, including its terminal element. In this fashion, we can either quickly reach the element $q \in P_e$ of r_i-type to which this copy of $x_{j,n}$ is attached, or we will see that $P_e \ncong \mathcal{A}$.

As we are building a punctual isomorphism from $\mathcal{A}$ to P_e, we may have already mapped r_i to some point q' in P_e. If $q \neq q'$, then our isomorphism will have failed. So we must have a strategy to ensure that this cannot occur if $P_e \cong \mathcal{A}$.

Another concern is what if our opponent shows some other copy of $x_{j,n}$ before we are ready for it. That is, we know that a copy of $x_{j,n}$ will eventually appear attached to almost every r_i, but what if our opponent shows a copy attached to their version of some r_i before we are ready to attach our own copy to that r_i? The same strategy will handle this concern.

6.6.1. *Leading chains.* For each r_i we have placed, we will define a *leading chain*, which is a completed chain in one of the components attached to r_i. The leading chain of an r_i will shift from stage to stage, but it will always have the following property:

> If $x_{j,n}$ is the leading chain for r_i at stage s, then it is the only copy of $x_{j,n}$ in the structure at stage s. That is, there is an $x_{j,n}$ attached to r_i, but not to any $r_{i'} \neq r_i$, and the $x_{j,n}$ attached to r_i was necessarily constructed by the chain method.

If $x_{j,n}$ is the leading chain for r_i at stage s, we say that C_j is *leading* for r_i at stage s. It follows that the copy of C_j attached to r_i at stage s is strictly longer than the copy attached to any other $r_{i'}$. Also, if $C_{j'}$ is leading for some $r_{i'}$ with $i \neq i'$, then $j \neq j'$.

We will obey the following rule with regards to leading chains and extending:

> When we extend a C_j attached to some r_i by adding new chains, we will only do this if C_j is not currently leading for any $r_{i'} \neq r_i$.

6.6.2. *The function K.* We will need an additional binary function K. This function will be symmetric and will only be nontrivially defined (that is, taking a value other than d) on pairs (y, z), where y is the initial element of some $x_{j,n}$, z is the initial element of some $x_{j',n'}$, $j \neq j'$, and y and z are both from components attached to the same r_i.

If y, z are initial elements from some chains attached to some r_i, and y', z' are the initial elements of the matching chains attached to some $r_{i'}$, then we will have $K(y, z) = K(y', z')$. We will thus abuse notation and write $K(x_{j,n}, x_{j',n'})$, where this is understood to mean the value of K on a pair of initial points of some copies of $x_{j,n}$ and $x_{j',n'}$ attached to the same r_i. When the value of $K(y, z)$ is not d, it will be the initial point of some label, constructed by the chain method.

6.6.3. *The strategy.* Suppose we are extending the copy of C_j attached to the same r_i as described in 6.5, and so j is not leading for any $r_{i'}$. At the stage we finish this extension, we will have added one or more chains to C_j, the largest having been built by the chain method, and the rest being copies of previously existing chains and added all at once. There may be some pairs of chains $(x_{j,n}, x_{j',n'})$ with $j \neq j'$ which now occur together attached to r_i, but which have never previously occurred attached to the some $r_{i'}$ (in particular, if r_i has a leading chain of type $x_{j',n'}$ with $j' \neq j$, then since the leading chain does not occur anywhere else in the structure, none of the newly added chains have previously occurred paired with it). For each of these pairs, $K(x_{j,n}, x_{j',n'})$ is not yet defined. We will immediately begin construction of a new label via the chain method, and for each of these pairs, we will define $K(x_{j,n}, x_{j',n'})$ to be the initial point of this new label (note that all the new pairs share the same label).

6.6.4. *Why it works.* Let us return to the situation where some $x_{j,n}$ has appeared in P_e, and we have located the point $q \in P_e$ of r_i-type to which it is attached. As our earlier strategy (6.4) will handle the case when q is a new point of r-type, we consider the situation where q is the image of some $r_{i'}$, but we have not yet attached a copy of $x_{j,n}$ to $r_{i'}$. Note that this encompasses both of the motivating concerns.

Let $C_{j'}$ be leading for $r_{i'}$. Since the copy of $C_{j'}$ attached to $r_{i'}$ is longer than any other copy of $C_{j'}$, and $x_{j,n}$ does not occur in it, it must be that $j \neq j'$. Fix $x_{j',n'}$ the leading chain of $r_{i'}$. We have already mapped $x_{j',n'}$ isomorphically into P_e, and its image is attached to q. Thus, in P_e, $x_{j,n}$ and $x_{j',n'}$ occur attached to the same element q, and so $K(x_{j,n}, x_{j',n'})$ must be defined in P_e (or we see that $P_e \ncong \mathcal{A}$). But the pair $(x_{j,n}, x_{j',n'})$ do not yet occur attached to the same element in $\mathcal{A}$, and so $K(x_{j,n}, x_{j',n'})$ is not yet defined in $\mathcal{A}$. When we eventually place copies of them together and define $K(x_{j,n}, x_{j',n'})$, the label it points to will be different from the label in P_e, by 6.3(6), and so we will have $P_e \ncong \mathcal{A}$.

6.6.5. *An important observation.* Note that when we have just placed an r_k by extending C_0, as described in 6.5.3, the only chains attached to r_k are of the form $x_{0,n}$. Thus there are no new pairs requiring the definition of K, and so we may skip creating a new label for this strategy when we do this.

6.7. **Shifting leading chains.** As we intend to grow each C_j to be infinite, but we are forbidden from extending C_j for r_i if C_j is leading for some $r_{i'} \neq r_i$, we must have a means of adjusting leading chains to make any given C_j no longer leading for any r_i. To help us achieve this, we will maintain the following at every stage s:

> For every $n > 0$, there are at most n numbers i such that we have placed r_i and $h(i, s) \leq n$.

Assuming the above promise holds at stage s, let j_0 be least such that C_{j_0} is not leading for any r_i. If $j_0 > 0$, then fix $i_0, i_1, \ldots, i_{j-1}$ such that C_k is leading for r_{i_k}, for all $k < j_0$. By our promise, there must be a $k < j_0$ with $h(i_k, s) \geq j_0$. By our initial description for $\mathcal{A}$, r_{i_k} is intended to have a copy of C_{j_0} attached to it.

We extend C_{j_0} for r_{i_k} as described in 6.5, and we build the label and define K as required in 6.6. As part of this process, we will have built some $x_{j_0,n}$ via the chain technique; we declare $x_{j_0,n}$ to be the new leading chain for r_{i_k}. We have now arranged that C_k is not leading for any r_i, and $k < j_0$. By repeating this, we will eventually reach a stage when C_0 is not leading for any r_i.

Having done this, if we wish to make C_j no longer leading for any r_i, fix the r_i for which C_j is currently leading. Since r_i is intended to have a copy of C_0 attached to it, we extend C_0 for r_i and build the required label, as described in 6.5 and 6.6, respectively. We then declare the $x_{0,n}$ which was just constructed via the chain method to be leading for r_i. We have now arranged that C_j is not leading for any r_i.

6.8. **The full construction.** Our construction proceeds by cycling between the following three phases:

(1) Placing the next r:
    (a) Let k be least such that we have not yet placed r_k.

   (b) We shift leading chains about (as described in 6.7) such that C_0 is not leading for any placed r_i.
   (c) We wait until a stage s when placing r_k would not violate the promise of 6.7. While we wait, we delay by building a chain not connected to any other part of the structure.
   (d) We place r_k by extending C_0, as described in 6.5.3. We declare $x_{0,n}$ to be the leading chain for r_k, where $x_{0,n}$ is the chain just constructed by the chain technique.
   (e) For each P_e that we are watching, this process will have mapped $x_{0,n}$ punctually isomorphically into P_e (or P_e will have proven itself not isomorphic to $\mathcal{A}$). From this we quickly obtain the element in P_e of r-type to which it is attached, and we declare that r_k maps to this element of P_e.
   (f) Meanwhile, we build the label for sets of size $k+1$, as required for 6.4.
(2) Extending all necessary C_j:
   (a) Let s_0 be the stage at which we enter this phase.
   (b) For each placed r_i and each j with $j \leq h(i, s_0)$, we do the following (one at a time):
      (i) We shift leading chains about (as described in 6.7) such that C_j is not leading for any placed r_i.
      (ii) We extend C_j for r_i (as described in 6.5).
      (iii) We build the label and define K as required for 6.6.
(3) Watching new P_e:
   (a) Let e be least such that we are not yet watching P_e. We declare that we are now also watching P_e.
   (b) For each placed r_i, we do the following (one at a time):
      (i) Let C_j be leading for r_i. We extend C_j for r_i (as described in 6.5).
      (ii) This process will have mapped some $x_{j,n}$ constructed by the chain technique punctually isomorphically into P_e (or P_e will have proven itself not isomorphic to $\mathcal{A}$). We obtain the element of r-type to which it is attached, and we declare that r_i maps to this element of P_e.
      (iii) Meanwhile, we build the label and define K as required for 6.6.

6.9. **Verification.** We must justify that the construction can proceed.

Claim 6.1. *Suppose we have placed $r_0, \ldots, r_{k-1}$. There is eventually a stage when placing r_k would not violate the promise of 6.7.*

Proof. Fix an s_0 sufficiently large such that for all $i \leq k$, if $\lim_s h(i, s) < \infty$, then $h(i, s_0) = \lim_s h(i, s)$, and such that if $\lim_s h(i, s) = \infty$, then $h(i, s_0) \geq k + 1$. We claim that s_0 is such a stage as we desire.

For $n \geq k + 1$, the promise is kept, as there are only $k + 1$ numbers $i \leq k$.

For $n < k + 1$, we know that there is at most one $i \leq k$ with $h(i, s_0) = m$, for each $1 \leq m \leq n$, and there is no number i with $h(i, s_0) = 0$. Thus there are at most n numbers $i \leq k$ with $h(i, s_0) \leq n$. □

Observe that since $h(i, s) \leq h(i, s + 1)$ for all i and s, if the promise of 6.7 holds at stage s_0 and we have not placed any new points r_i between stages s_0 and s, then the promise must also hold at stage s. It follows that the promise is kept at every stage. Thus the construction can proceed.

That $\mathcal{A}$ is punctual follows from construction. We are always placing new points in the structure, and we always define each function on each tuple as soon as that tuple occurs in the structure.

We have already argued throughout the construction why $\mathcal{A}$ is punctually categorical, modulo the black box of the chain technique. Once we go through the phase for declaring P_e watched, we will have mapped each of the existing r_i and each of their leading chains. Henceforth we will always maintain that the leading chain of each r_i is mapped into P_e, which lets us apply the arguments of 6.6 and 6.4.

Note that a finite fragment of $\mathcal{A}$ will never be mapped to P_e by our construction, namely any delaying chains created before we have begun watching P_e. Also, any of the finitely many instances of an $x_{j,n}$ which were created before we began watching P_e will not be mapped until their corresponding copy of C_j is extended, which may not happen for a long time. The former is corrected nonuniformly, while the latter can be accommodated by adding a sufficiently large constant to our primitive recursive time bound for the convergence of the punctual isomorphism.

The construction of $\mathcal{B}$ is as initially described, with the obvious changes to incorporate the additional elements of the structure we subsequently introduced.

This completes the proof.

7. Proof of Theorem 1.5

In light of Theorem 1.4, it suffices to construct a punctually categorical structure $\mathcal{A}$ in a unary language and a computable $\mathcal{B} \cong \mathcal{A}$ such that every isomorphism between $\mathcal{A}$ and $\mathcal{B}$ is of PA$(0')$ degree. Fix X_0 and X_1 disjoint Σ^0_2 sets such that every separator is of PA$(0')$ degree. We shall arrange $\mathcal{A}$ and $\mathcal{B}$ such that every isomorphism between them computes a separator.

Fix primitive recursive predicates φ_0, φ_1 such that for all $n < \omega$ and each $i \in \{0,1\}$, $n \in X_i \iff \exists^{<\infty} s\, \varphi_i(n,s)$.

As in the construction from the previous section, we will use a special element d to "dump" all otherwise unspecified function values. As we do not have constants in our language, we will instead have a unary function d such that $d(x) = d$ for all $x \in \mathcal{A}$.

We will also make use of the chain construction described in 6.3. Recall that this technique uses only unary functions. We will again use chains to delay, among other uses.

7.1. **The naïve coding strategy.** We must code X_0 and X_1's behavior on n. In the notation of the previous theorem, the coding module will consist of two fragments of C_n. For example, at a stage s, our fragments could look as follows:

$$x_{n,0} \longleftarrow x_{n,1} \longleftarrow x_{n,2}$$

$$x_{n,0} \longleftarrow x_{n,1}$$

We wait until a stage s_1 when $\varphi_1(n, s_1)$ holds, and then we extend the lower fragment:

$$x_{n,0} \longleftarrow x_{n,1} \longleftarrow x_{n,2}$$

$$x_{n,0} \longleftarrow x_{n,1} \longleftarrow x_{n,2} \longleftarrow x_{n,3}$$

We then wait until a later stage s_2 when $\varphi_0(n, s_2)$ holds, and we extend the upper fragment:

$$x_{n,0} \longleftarrow x_{n,1} \longleftarrow x_{n,2} \longleftarrow x_{n,3} \longleftarrow x_{n,4}$$

$$x_{n,0} \longleftarrow x_{n,1} \longleftarrow x_{n,2} \longleftarrow x_{n,3}$$

We continue in this fashion, always extending the shorter fragment, and only extending the upper fragment when $\varphi_0(n, s)$ holds, and only extending the lower when $\varphi_1(n, s)$ holds.

In $\mathcal{B}$, the respective location will be isomorphic, but we will always keep the lower fragment the longer of the two. If $n \in X_0$, and thus $n \notin X_1$, then there will be only finitely many s for which $\varphi_0(n, s)$ holds, but there will be infinitely many s with $\varphi_1(n, s)$ holding. Thus there will come a point where we extend the lower fragment in $\mathcal{A}$ and then never again extend, so at the end of the construction, the lower fragment in $\mathcal{A}$ is the longer of the two. So any isomorphism from $\mathcal{A}$ to $\mathcal{B}$ must map the upper fragment in $\mathcal{A}$ to the upper fragment in $\mathcal{B}$. Similarly, if $n \in X_1$ and thus not in X_0, then at the end of the construction, the upper fragment in $\mathcal{A}$ is the longer of the two, and so any isomorphism from $\mathcal{A}$ to $\mathcal{B}$ must map the upper fragment to the lower fragment. If $n \notin X_0 \cup X_1$, then there will be infinitely many s with $\varphi_0(n, s)$ and infinitely many s with $\varphi_1(n, s)$, so we will extend both fragments infinitely and create two copies of C_n. So either mapping will be correct in this case. Thus we will be able to read the value of a separator at n from whether an isomorphism sends the top fragment of $\mathcal{A}$ to the top or bottom fragment in $\mathcal{B}$.

The technique for making $\mathcal{B}$ computable is as follows: whenever $\mathcal{A}$ begins the extension process for one of the fragments, $\mathcal{B}$ simply observes without matching $\mathcal{A}$ until the process is complete, including the construction of the auxiliary labels we will describe shortly. Then $\mathcal{B}$ extends its top fragment to match the longer of $\mathcal{A}$'s two fragments, and it extends its bottom fragment to match the lower of the two.

Remark 7.1. *Our coding location does not have a point r_i at the base of the C_n, as was done in the previous proof. This is crucial, as if we placed a sequence $(r_i)_{i\in\omega}$ like that, we would not have access to the binary function f from 6.4 to use to prevent our opponent from placing the sequence faster than we do.*

7.2. **The two dangers.** Since we are restricted to unary functions, we do not have the function K from 6.6. We must find a different way to address the dangers K was previously used to prevent. There are two such dangers, and they must be treated separately, albeit similarly. We explain them both first, and then we will go on to explain our strategies for handling them. Suppose P_e is a punctual structure we are watching, and we are attempting to build a punctual isomorphism from $\mathcal{A}$ to P_e.

To illustrate the first danger, suppose again that our coding location for n currently looks like this:

$$\begin{array}{l} x_{n,0} \longleftarrow x_{n,1} \longleftarrow x_{n,2} \\ \\ x_{n,0} \longleftarrow x_{n,1} \end{array}$$

Let us suppose that we have mapped this punctually isomorphically into P_e. Since it is possible that we will eventually extend the lower fragment, adding (among other things) a copy of x_2, P_e may grow impatient and place a copy of x_2 in the lower fragment before we do so, so that in P_e the coding location looks like this:

$$\begin{array}{l} x_{n,0} \longleftarrow x_{n,1} \longleftarrow x_{n,2} \\ \\ x_{n,0} \longleftarrow x_{n,1} \longleftarrow x_{n,2} \end{array}$$

We have nowhere in $\mathcal{A}$ to map to this new copy of $x_{n,2}$, so we cannot extend our isomorphism punctually. We would like instead to punish P_e for its audacity by ensuring that $P_e \ncong \mathcal{A}$. One way to do this would be to never again extend the lower fragment, but that would kill our attempt at coding.

To illustrate the second danger, suppose we are extending one of the fragments of the coding location. Again, we have mapped the already existing fragments punctually isomorphically into P_e. Our extension procedure is to extend the shorter fragment by two chains. P_e might decide to take this opportunity to break our isomorphism by instead extending both fragments by one chain:

$$\begin{array}{ccc}
\mathcal{A} & & P_e \\
\\
x_{n,0} \longleftarrow x_{n,1} \longleftarrow x_{n,2} & & x_{n,0} \longleftarrow x_{n,1} \longleftarrow x_{n,2} \\
\\
x_{n,0} \longleftarrow x_{n,1} & & x_{n,0} \longleftarrow x_{n,1} \\
\Downarrow & & \Downarrow \\
x_{n,0} \longleftarrow x_{n,1} \longleftarrow x_{n,2} & & x_{n,0} \longleftarrow x_{n,1} \longleftarrow x_{n,2} \qquad x_{n,3} \\
\\
x_{n,0} \longleftarrow x_{n,1} \qquad\qquad x_{n,3} & & x_{n,0} \longleftarrow x_{n,1} \\
\Downarrow & & \Downarrow \\
x_{n,0} \longleftarrow x_{n,1} \longleftarrow x_{n,2} & & x_{n,0} \longleftarrow x_{n,1} \longleftarrow x_{n,2} \longleftarrow x_{n,3} \\
\\
x_{n,0} \longleftarrow x_{n,1} \longleftarrow x_{n,2} \longleftarrow x_{n,3} & & x_{n,0} \longleftarrow x_{n,1} \longleftarrow x_{n,2}
\end{array}$$

Again, if P_e does this, we wish to ensure that $P_e \ncong \mathcal{A}$.

7.3. **Solving the first danger.** Each chain $x_{n,j}$ will point (via some unary function) from its terminal node to some label. This label will be a chain, and the first copy of it will be created by the chain technique immediately after the first copy of $x_{n,j}$ is placed. If the coding strategy acts enough that both fragments of C_n have a copy of $x_{n,j}$, then the labels to which the two $x_{n,j}$ point will isomorphic, and they may even be the same label. We will make that decision when we place the second copy of $x_{n,j}$, and our default will be to make them point to the same label.

However, if P_e were to show us part of a second $x_{n,j}$ before we have placed it, we can delay ourselves (extend the current chain construction, or begin a new delaying chain if necessary) until P_e shows us the entirety of this $x_{n,j}$ (or proves itself non isomorphic to $\mathcal{A}$). When P_e finishes this second $x_{n,j}$, it must decide whether to map the terminal element of it to the same label as its first copy of $x_{n,j}$, or whether to map it to a separate copy. Whichever choice it makes, we will make the opposite if and when we ever place our second $x_{n,j}$.

Note that if our coding strategy never calls upon us to place a second copy of $x_{n,j}$, then P_e is still non-isomorphic, because it has two copies and we only one. Also, if we are called upon to place a second copy of the label, we delay the extension until a stage where this is a small number of points, and then we will place the entire label at once, just as we simultaneously place the entire copy of $x_{n,j}$ at once.

For some other $P_{e'}$, we are able to map this second copy of the label punctually for the same reason we are able to map the second copy of $x_{n,j}$ punctually: we first build the first copy of $x_{n,j+1}$, according to the chain method, which maps it punctually into $P_{e'}$, and then from the terminal element of $x_{n,j+1}$ in $P_{e'}$ we are able to quickly generate the entire copy of $x_{n,j}$ and the label (since the number of points involved is small relative to the current stage).

7.3.1. *Handling multiple P_e.* The above discussion is sufficient if we are only concerned with a single P_e, but we will be considering several punctual structures at once. It may be that several of them have rushed the placement of the second $x_{n,j}$, and some of them decided to make it point to the same copy of the label, while others decided to make it point to a separate copy. We obviously cannot make the opposite choice as all of them.

We instead defeat the highest priority P_e which has not already been proven non-isomorphic, and for all lower priority structures which rushed the placement but made the other choice, we will restart the construction of their punctual isomorphism. Our construction of the isomorphism for a given P_e can only be injured in this fashion at most e times (as each $P_{e'}$ with $e' < e$ which causes such an injury is henceforth known to be non-isomorphic), and so the isomorphism to P_e is only restarted finitely many times, after which P_e can never rush the placement of $x_{n,j}$ without being proven non-isomorphic itself.

7.4. **Solving the second danger.** For the moment pretend we are not implementing our solution to the first danger. We will see how to integrate the two solutions later.

Each chain $x_{n,j}$ will point (via some unary function) from its terminal node to some label. This label will be a chain, and the first copy of it will be created by the chain technique immediately after the first copy of $x_{n,j}$ is placed. If there are two copies of $x_{n,j}$, they will point to separate by isomorphic copies of the same label. Each chain $x_{n,j+1}$ will also point (via some unary function) from its *initial* node to some label. This label will be isomorphic to the label which $x_{n,j}$ points to with its terminal node, and the $x_{n,j}$ and $x_{n,j+1}$ from the same fragment may even point to the same label (the first with its terminal node and the second with its initial node). Whichever choice we make for one fragment, we will make the same choice for the other fragment (if and when the other fragment gains its own copy of $x_{n,j+1}$).

Note that we do not need to make this choice until we finish the construction of $x_{n,j+1}$. This is because until $x_{n,j+1}$ is complete, we have not created the second copy of $x_{n,j}$ which is to go in the same fragment as it. On the other hand, if our opponent is attempting to diagonalize against us as described for the second danger, they must make this choice as soon as they *begin* their copy of $x_{n,j+1}$. This is because they will have placed the initial node of $x_{n,j+1}$, and the fragment they intend it for already has a copy of $x_{n,j}$. Whichever choice they make, we will make the opposite.

More precisely, while we are building $x_{n,j+1}$ via the chain technique, we are simultaneously mapping it punctually into P_e. We have already mapped our (currently solitary) copy of $x_{n,j}$ into P_e. In P_e, we check whether the initial node of $x_{n,j+1}$ maps to the same element as the terminal node of $x_{n,j}$. If so, when we finish the chain construction, we make $x_{n,j}$ and $x_{n,j+1}$ point to different copies of the label. If not, we make them point to the same copy. Then if P_e does attempt the diagonalization described for the second danger, we will have proven $P_e \ncong \mathcal{A}$.

7.4.1. *Handling multiple P_e.* Since P_e will in general be slower than $\mathcal{A}$ (by a primitive recursive factor), we can expect to complete $x_{n,j+1}$ before P_e does, and thus we will make the appropriate choice about the labels before P_e finishes its copy of $x_{n,j+1}$. Fortunately, we only needed to see the first element of P_e's copy in order to make our decision. However, this means that we may make our choice before we know if P_e intends to diagonalize in the manner we fear.

If in P_e the two chains point to the same label, then the situation is unambiguous: P_e must intend to diagonalize, and our action has turned the tables and proven that $P_e \ncong \mathcal{A}$. If, however, P_e had the two chains point to separate copies of the label, then it may be because P_e intends to follow us faithfully and put the chains in separate fragments. This is good for our construction of an isomorphism, but it means we will not have proven P_e non-isomorphic. In this situation, we cannot restart the isomorphism construction for lower priority structures—it might lead to P_0 injuring lower priority constructions infinitely many times.

We solve this by using a sequence of labels $\ell_{n+j}, \dots, \ell_0$. The terminal node of $x_{n,j}$ will point to the initial node ℓ_{n+j}, and the terminal node of each ℓ_{i+1} will point to the initial node of ℓ_i, until ℓ_0 is reached. As soon as we finish building the first copy of $x_{n,j}$, we will begin building these labels in order of decreasing subscript, using the chain technique for each.

The initial node of $x_{n,j+1}$ will point to an isomorphic sequence. This sequence will merge with the sequence attached to the $x_{n,j}$ from the same fragment. That is, there will be a $k \in [0, n+j]$ such that for all $i > k$, the ℓ_i label reached from the terminal node of $x_{n,j}$ is separate from the copy of the ℓ_i label reached from the initial node of $x_{n,j+1}$, and for all $i \leq k$, the ℓ_i label reached from the terminal node of $x_{n,j}$ is the same as the ℓ_i label reached from the initial node of $x_{n,j+1}$. Note that achieving this simply requires placing copies of ℓ_i for $i > k$ as we place the initial node of $x_{n,j+1}$, and then making the terminal node of ℓ_{k+1} (or the initial node of $x_{n,j+1}$ for $k = n+j$) point to the initial node of the already existing ℓ_k.

$$
\begin{array}{l}
\ell_0 \\
\uparrow \\
\vdots \\
\uparrow \\
\ell_k \longleftarrow \ell_{k+1} \longleftarrow \cdots \longleftarrow \ell_{n+j} \longleftarrow x_{n,j+1} \\
\uparrow \\
\ell_{k+1} \\
\uparrow \\
\vdots \\
\uparrow \\
\ell_{n+j} \\
\uparrow \\
x_{n,j}
\end{array}
$$

Again, we must choose k when we finish the construction of the first copy of $x_{n,j+1}$.

By the time we begin building the first copy $x_{n,j+1}$, this entire sequence of labels will be small relative to the current stage (as usual, we can delay until this is so). For each $e < n + j$ which we are watching and for which we have not already proven $P_e \ncong \mathcal{A}$, as we build the first copy of $x_{n,j+1}$, we will map it punctually isomorphically into P_e, and so we will have its initial node to hand. We will check whether the ℓ_{e+1} label reached from the terminal node of P_e's copy of $x_{n,j}$ is the same as the ℓ_{e+1} label reached from the initial node of $x_{n,j+1}$. We let $k = e$ for the least such e which makes the ℓ_{e+1} labels the same, or $k = n + j$ if there is no such e.

Now, suppose some P_e attempts to diagonalize in the described fashion. If $e \geq n + j$, we simply restart the construction of the punctual isomorphism for P_e; this sort of injury can happen at most finitely many times. If it gives its $x_{n,j}$ and $x_{n,j+1}$ the same copy of ℓ_{e+1}, then $k \leq e$, and so P_e made the wrong choice (and is now proven non-isomorphic). If it gives its $x_{n,j}$ and $x_{n,j+1}$ separate copies, then either $k > e$, or some $e' < e$ is being proven non-isomorphic. In the former case, P_e proves itself non-isomorphic. In the latter case, we restart the construction of the isomorphism for P_e; since the particular e' will never be considered again, this sort of injury can happen at most finitely many times.

7.5. **Integrating the strategies for the two dangers.** The final description above of our strategy for handling the second danger requires no further modification. As described in that strategy, the terminal node of each $x_{n,j}$ will point to a sequence of labels $\ell_{n+j} \to \ell_{n,j-1} \to \cdots \to \ell_0$. The reader may notice that that earlier strategy did not make use of ℓ_0. We will use ℓ_0 to ward off the first danger, by implementing the previously described strategy with it.

If both fragments of C_n contain a copy of $x_{n,j}$, then the $\ell_{n+j}, \ldots, \ell_1$ which their terminal nodes point to will be distinct (though identical). The ℓ_0 which they point to may be the same, however. The strategy for whether to make them the same or not is as described in 7.3.

7.6. **Running the construction.** The construction cycles through the following phases:

(1) Updating coding locations:
   (a) Let s_0 be the stage we begin this phase.
   (b) For every $n < s_0$ such that we have previously extended one of the C_n fragments:
      (i) Let $s_1 < s_0$ be the last time we extended one of the C_n fragments.
      (ii) Let $i = 0$ if we extended the lower C_n fragment at stage s_1, and let $i = 1$ otherwise.
      (iii) If there is a $t \in [s, s_0)$ with $\varphi_i(n,t)$ holding, extend the shorter of the C_n fragments. Build all appropriate labels according to the previously described strategies.
   (c) For every $n < s_0$ such that both C_n fragments are empty, extend the upper fragment (that is, build $x_{n,0}$ and all appropriate labels).
(2) Declaring new P_e watched.
   (a) Let e be least such that we have not already declared P_e to be watched. Declare that we are now watching P_e.

The verification that $\mathcal{A}$ is punctual and punctually categorical is routine. Note that again there is finitely much of $\mathcal{A}$ on which our punctual isomorphism with P_e will never be defined, namely the delaying chains constructed before the final time we began building the isomorphism, and also any C_n fragments that never extended after the final time we began building the isomorphism. Also, there is finitely much of $\mathcal{A}$ such that the map is only defined on it very late, as any C_n fragments constructed before the final time we began building the isomorphism will not be mapped until they are next extended. The former is handled nonuniformly, while the latter is handled by adding a sufficiently large constant to the primitive recursive bound on our isomorphism's convergence.

This completes the proof of Theorem 1.5.

8. Computable relational structures as automorphism bases

Theorem 8.1. *For every computable structure $\mathcal{C}$ in a finite relational language L, there is a punctual, punctually categorical structure $\mathcal{A}$ in a finite language $L' \supset L$ and a quantifier-free formula $\varphi(y)$ in the language L' such that:*

(1) The reduct of $\mathcal{A}_\varphi = \{y \in \mathcal{A} : \mathcal{A} \models \varphi(y)\}$ to L is computably isomorphic to $\mathcal{C}$; and
(2) The reduct of $\mathcal{A}_\varphi$ to L is an automorphism base for $\mathcal{A}$.

Further, for every computable $\mathcal{D} \cong \mathcal{C}$, there is a computable $\mathcal{B} \cong \mathcal{A}$ such that the reduct of $\mathcal{B}_\varphi$ to L is computably isomorphic to $\mathcal{D}$.

Proof. This is only a small modification of the proof of Theorem 1.6. We will also incorporate a technique used in the proof Theorem 1.5.

$L' \setminus L$ will consist of all the symbols used in the proof of Theorem 1.6, as well as relation symbols $(S_R)_{R \in L}$. Each S_R will have the same arity as R. In place of a sequence of indiscernibles $(r_i)_{i\in\omega}$, we will have $(r_a)_{a\in\mathcal{C}}$. In the notation of the proof of Theorem 1.6, each r_a will have an attached copy of C_i for every $i < \omega$. For each $R \in L$ and $a_0, \ldots, a_{n-1} \in \mathcal{C}$, we will have $\mathcal{A} \models R(r_{a_0}, \ldots, r_{a_{n-1}})$ iff $\mathcal{C} \models R(a_0, \ldots, a_{n-1})$. The fact that $\mathcal{C}$ is only computable, rather than punctual, is not a concern for this—our construction lets us delay placing the next r_a for as long as we like, so we can delay until $\mathcal{C}$ has converged on all tuples we need to know before placing r_a.

Our default position will be that if we do not otherwise specify the value of a relation on a given tuple, then it is false.

The set $\{r_a : a \in \mathcal{C}\}$ is our $\mathcal{A}_\varphi$, its reduct to L is isomorphic to $\mathcal{C}$ via the computable isomorphism $a \mapsto r_a$, and it is defined by the formula $\varphi(y) : b_2(y) = y \wedge y \neq d$, as in 6.4.3. In general, the construction proceeds as the proof for Theorem 1.6. There is only one new element required.

Suppose we are monitoring P_e, we have already placed elements $r_{a_0}, \ldots, r_{a_{n-1}}$, and we are about to place r_y. Thus, for each $i < n$, we have already mapped r_{a_i} to some $z_i \in P_e$, and we have also mapped r_{a_i}'s leading chain x_i to some chain w_i in P_e. We assume also that for all $R \in L$ and all $i_0, \ldots, i_k < n$,

$$\mathcal{C} \models R(a_{i_0}, \ldots, a_{i_k}) \iff \mathcal{A} \models R(r_{a_{i_0}}, \ldots, r_{a_{i_k}}) \iff P_e \models R(z_{i_0}, \ldots, z_{i_k}).$$

When we place r_y (which we do by extending its C_0, as explained in 6.5.3), we will punctually isomorphically map its leading chain x_y to some chain w_y in P_e, which will point to some $z_y \in P_e$ of r-type, and we will map r_y to z_y. Our concern is that P_e might not respect the language L on r_y. That is, there may be some $R \in L$ and some $i_0, \ldots, i_{k-1} < n$ such that

$$\mathcal{C} \models R(a_{i_0}, \ldots, a_{i_{k-1}}, y) \iff \mathcal{A} \models R(r_{a_{i_0}}, \ldots, r_{a_{i_{k-1}}}, r_y) \iff P_e \models \neg R(z_{i_0}, \ldots, z_{i_{k-1}}, z_y).$$

Without loss of generality, assume this occurs with $P_e \models \neg R(z_{i_0}, \ldots, z_{i_{k-1}}, z_y)$. Then notice that the chains $x_{i_0}, \ldots, x_{i_{k-1}}, x_y$, being leading, currently occur only once each in $\mathcal{A}$, and the elements of r-type to which they point $(r_{a_{i_0}}, \ldots, r_{a_{i_{k-1}}}, r_y)$ satisfy R. The isomorphic chains $w_{i_0}, \ldots, w_{i_{k-1}}, w_y$, on the other hand, point to elements of r-type which do not satisfy R. Thus this is a configuration which occurs in P_e but not in $\mathcal{A}$.

Let $b_{i_0}, \ldots, b_{i_{k-1}}, b_y$ be the initial nodes of the chains $w_{i_0}, \ldots, w_{i_{k-1}}, w_y$, respectively. We will check if $P_e \models S_R(b_{i_0}, \ldots, b_{i_{k-1}}, b_y)$. Whichever choice P_e makes, when we eventually place copies of these chains in $\mathcal{A}$ pointing to r-type elements which do not satisfy R, we will make the opposite choice. We will maintain this rule: for any copies of $x_{i_0}, \ldots, x_{i_{k-1}}, x_y$ we place in $\mathcal{A}$, when they point to r-type elements which do not satisfy R, then the value of S_R on their initial nodes will be the opposite of the choice P_e originally made. Thus we will have proven that $P_e \not\cong \mathcal{A}$.

Note that a rule introduced in this fashion at stage s_0 will have no overlap with a rule introduced in this fashion at stage $s_1 > s_0$, because at least one of the leading chains involved will be different, namely the leading chain of the new r_y. So they will not interfere with each other.

This becomes slightly more complicated when there are P_e. It may be that several punctual structures we are watching violate the same R with the same tuple of elements, but they do not all choose the same value for S_R. In this situation, we do as was done in 7.3, and we act to defeat the highest priority P_e and restart the construction of the isomorphism for later $P_{e'}$. Since P_e has now been proven non-isomorphic, and so can never again cause injury, no isomorphism construction is restarted more than finitely many times.

This describes the construction of $\mathcal{A}$.

Now, suppose $\mathcal{D} \cong \mathcal{C}$ is computable. We will build $\mathcal{B} \cong \mathcal{A}$ with elements $(r_b)_{b\in\mathcal{D}}$ such that $b \mapsto r_b$ is an isomorphism from $\mathcal{D}$ to the reduct of $\mathcal{B}_\varphi$ to L. To build $\mathcal{B}$ computably, fix a computable increasing sequence of finite structures $(D_s)_{s\in\omega}$ with $\mathcal{D} = \bigcup_s D_s$ and $D_0 = \emptyset$. For each $s+1$, we can wait until we see a substructure $C_{s+1} \subset \mathcal{C}$ with $D_{s+1} \cong C_{s+1}$, and then wait further until all of the r_a for $a \in C_{s+1}$ have been placed in $\mathcal{A}$. Then we can place r_b for $b \in D_{s+1} \setminus D_s$, copying the appropriate r_a. □

Corollary 8.2. *There is a punctual, punctually categorical structure of Scott Rank* $\omega_1^{ck}+1$.

Proof. Fix a computable structure $\mathcal{C}$ in a finite relational language which is of Scott Rank $\omega_1^{ck}+1$, e.g. the Harrison ordering. By the Scott Analysis, the automorphism group of $\mathcal{C}$ has a non-empty neighborhood with no Δ^1_1 element. Let $\mathcal{A}$ be as in Theorem 8.1. Since the reduct of $\mathcal{A}_\varphi$ to the language of $\mathcal{C}$ is a computable structure computably isomorphic to $\mathcal{C}$, and it is an automorphism base for $\mathcal{A}$, it follows that the the automorphism group of $\mathcal{A}$ has a non-empty neighborhood with no Δ^1_1 element, and thus that $\mathcal{A}$ has Scott rank $\omega_1^{ck}+1$. □

Corollary 8.3. *There is a punctual, punctually categorical structure which is not* Δ^1_1*-categorical.*

Proof. Fix computable structures $\mathcal{C}$ and $\mathcal{D}$ in a finite relational language which are isomorphic but have no Δ^1_1-isomorphism, e.g. two appropriate presentations of the Harrison ordering. Let $\mathcal{A}$ and $\mathcal{B}$ be as in Theorem 8.1. Any isomorphism Δ^1_1 isomorphism from $\mathcal{A}$ to $\mathcal{B}$ would induce a Δ^1_1 isomorphism from $\mathcal{A}_\varphi$ to $\mathcal{B}_\varphi$, and thus from $\mathcal{C}$ to $\mathcal{D}$. □

Remark 8.4. Matthew Harrison-Trainor has suggested to view the proof above as an effective functor from computable relational structures in a finite signature to punctually categorical structures. Although the "functor" depends on the enumeration of $\mathcal{C}$ and the uniform total enumeration of all punctual structures, the dependency is restricted only to the isomorphism types of the complex labels that we use to code $\mathcal{C}$ within $\mathcal{A}$ and the values of the various relations S_R.

References

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

[BDKM19] Nikolay Bazhenov, Rod Downey, Iskander Kalimullin, and Alexander Melnikov. Foundations of online structure theory. *Bull. Symb. Log.*, 25(2):141–181, 2019.

[BFKMn17] N. A. Bazhenov, A. N. Frolov, I. Sh. Kalimullin, and A. G. Mel′ nikov. Computability of distributive lattices. *Sibirsk. Mat. Zh.*, 58(6):1236–1251, 2017.

[BHTK+19] Nikolay Bazhenov, Matthew Harrison-Trainor, Iskander Kalimullin, Alexander Melnikov, and Keng Meng Ng. Automatic and polynomial-time algebraic structures. *The Journal of Symbolic Logic*, pages 1–32, 04 2019.

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

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

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

[DHK03] R. Downey, D. Hirschfeldt, and B. Khoussainov. Uniformity in the theory of computable structures. *Algebra Logika*, 42(5):566–593, 637, 2003.

[DHTK+] R. Downey, M. Harrison-Trainor, I. Kalimullin, A. Melnikov, and D. Turetsky. Graphs are not universal for online computability. Preprint.

[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.

[DMN] Rod Downey, Alexander Melnikov, and Keng Meng Ng. Foundations of online structure theory ii: the operator approach. *Preprint.*

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

[GMR89] S. S. Goncharov, A. V. Molokov, and N. S. Romanovskiĭ. Nilpotent groups of finite algorithmic dimension. *Sibirsk. Mat. Zh.*, 30(1):82–88, 1989.

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

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

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

[Gro07] Misha Gromov. *Metric structures for Riemannian and non-Riemannian spaces*. Modern Birkhäuser Classics. Birkhäuser Boston Inc., Boston, MA, english edition, 2007. Based on the 1981 French original, With appendices by M. Katz, P. Pansu and S. Semmes, Translated from the French by Sean Michael Bates.

[Hir17] Denis R. Hirschfeldt. Some questions in computable mathematics. In *Computability and complexity*, volume 10010 of *Lecture Notes in Comput. Sci.*, pages 22–55. Springer, Cham, 2017.

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

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

[Kie98] H. A. Kierstead. Recursive and on-line graph coloring. In Yu. L. Ershov, S. S. Goncharov, A. Nerode, and J. B. Remmel, editors, *Handbook of recursive mathematics, Vol. 2*, volume 139 of *Stud. Logic Found. Math.*, pages 1233–1269. North-Holland, Amsterdam, 1998.

[KM19] iskander Kalimullin and Russell Miller. Primitive recursive fields and categoricity. *Alg.Log.*, 58(1):132–138, 2019.

[KMM19] Iskander Kalimullin, Alexander Melnikov, and Antonio Montalbán. Definability and punctual computability. Preprint., 2019.

[KMN17] Iskander Kalimullin, Alexander Melnikov, and Keng Meng Ng. Algebraic structures computable without delay. *Theoret. Comput. Sci.*, 674:73–98, 2017.

[KN08] Bakhadyr Khoussainov and Anil Nerode. Open questions in the theory of automatic structures. *Bull. Eur. Assoc. Theor. Comput. Sci. EATCS*, (94):181–204, 2008.

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

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

[MN13] Alexander G. Melnikov and André Nies. The classification problem for compact computable metric spaces. In Paola Bonizzoni, Vasco Brattka, and Benedikt Löwe, editors, *The Nature of Computation. Logic, Algorithms, Applications*, pages 320–328, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

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

School of Mathematics and Statistics, Victoria University of Wellington, PO Box 600, Wellington, New Zealand.

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

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