PRIMITIVE RECURSIVE CATEGORICITY SPECTRA OF FUNCTIONAL STRUCTURES

NIKOLAY BAZHENOV, HEER TERN KOH, AND KENG MENG NG

Abstract. For the notion of degree of categoricity, we study an analogous notion for punctual structures. We show that such notions coincide for non- Δ^0_1-categorical injection structures, and construct an example of a Δ^0_1-categorical injection structure for which these notions differ. Additionally, we also show that in every non-zero c.e. Turing degree, there exists a PR-degree that is low for punctual isomorphism (to be defined), and also a PR-degree that is a degree of punctual categoricity.

§1. Introduction. One of the key research programs in computable structure theory is the investigation of the algorithmic content of isomorphisms between computable structures (first defined in [29, 31]). More specifically, given some computable structure $\mathcal{A}$, researchers attempt to find a Turing degree $\mathbf{d}$ so that for any two computable presentations $\mathcal{B}, \mathcal{C} \cong \mathcal{A}$, $\mathbf{d}$ computes an isomorphism between $\mathcal{B}$ and $\mathcal{C}$. For such a degree $\mathbf{d}$ and structure $\mathcal{A}$, we say that $\mathcal{A}$ is $\mathbf{d}$-*computably categorical*. Intuitively, this is a measure of the amount of non-computable information required in order to algorithmically produce isomorphisms between copies of $\mathcal{A}$. Under such a setting, the algorithmic content of isomorphisms on various classes of structures have been studied (see, e.g., [1, 7, 8, 10, 21, 23, 30, 32, 33]).

Conversely, given a Turing degree $\mathbf{d}$, one can also ask if such a degree is able to compute isomorphisms between certain structures. To make such a question meaningful, one should really consider only the structures $\mathcal{A}$ for which $\mathbf{d}$ is the least degree that is capable of producing isomorphisms of $\mathcal{A}$. Otherwise, the answer is always trivially yes, as there exist structures that are *computably categorical*; structures which have computable isomorphisms between any two of their presentations. In the event that a structure $\mathcal{A}$ exists for which $\mathbf{d}$ is the least degree that computes an isomorphism between any two given presentations of $\mathcal{A}$, then we say that $\mathbf{d}$ is a *degree of categoricity* [19, 22]. We refer to, e.g., [5, 6, 13–17, 19] for known results on degrees of categoricity. On the other hand, if $\mathbf{d}$ is such that any structure that is $\mathbf{d}$-computably categorical is also computably categorical, then we say that $\mathbf{d}$ is *low for computable isomorphism* [20]. Intuitively, this means that $\mathbf{d}$ is 'computationally weak' in the sense that it provides no additional information when being used to compute isomorphisms.

Received March 10, 2026 UTC.
2020 *Mathematics Subject Classification.* Primary 03C57, 03D45, 03D55.
Key words and phrases. punctual structures, degrees of categoricity.

© The Author(s), 2026. Published by Cambridge University Press on behalf of The Association for Symbolic Logic.
0022-4812/00/0000-0000
DOI:10.1017/jsl.2026.10230

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

Closely related to computable structure theory is the study of *punctual structures*. These are structures with domain ω and whose operations and relations are uniformly primitive recursive [26]. The study of such structures was motivated by the phenomenon that 'feasible' presentations may be extracted from computable ones simply by eliminating the use of unbounded search [9, 11, 12, 24]. Thus, suggesting that isolating the minimisation operator provides insight into the gap between 'feasible' presentations and computable ones. Such an approach has attracted a fair amount of attention and has proven to provide a different perspective from computable structure theory (see [2, 18] for surveys regarding punctual structures). Following this pattern, we study the analogue of degrees of categoricity in the present article.

1.1. Preliminaries. Recall that the *degree of categoricity* of a computable structure is the least Turing degree $\mathbf{d}$ which computes an isomorphism between any two computable presentations of the structure. Evidently, to show that some structure has degree of categoricity $\mathbf{d}$, one must first show that $\mathbf{d}$ is sufficient to produce isomorphisms between any two computable presentations of the given structure. Second, to show that such a degree is 'sharp', one typically constructs two computable presentations so that any isomorphism f between these presentations is such that $f \geq_T \mathbf{d}$. Roughly speaking, the degree of categoricity of a computable structure measures how algorithmically complex isomorphisms between computable presentations of the given structure are. In order to study the primitive recursive analogue of such notions, we require the following definition.

DEFINITION 1.1 [25]. Let $f, g : \omega \to \omega$ be total functions. f reduces to g, written $f \leq_{PR} g$ if there is some primitive recursive scheme Ψ such that $\Psi^g = f$. If we further have that $g \leq_{PR} f$, then we write $f \equiv_{PR} g$. The induced degree structure shall be referred to as the PR-degrees.

Just like how Turing degrees are a measure of non-computability, the PR-degrees analogously measures the primitive recursive content of various functions. Intuitively, if $f \leq_{PR} g$, then g grows faster than f does, and is thus more 'primitive recursively complex' than f. Under this setting, and following the ideas and terminology from computable structure theory, we define the following definition.

DEFINITION 1.2 [3]. For a given punctual structure A, we use $\mathtt{PRCatSpec}(A)$ to denote the set containing all PR-degrees $\mathbf{d}$ such that for any punctual structure $B \cong A$, there is an isomorphism $f : A \to B$ and $\mathbf{d} \geq_{PR} f, f^{-1}$.

Since the inverse of a primitive recursive function is not necessarily primitive recursive, it follows that for a given function f, f and f^{-1} may belong to different PR-degrees. As such, when defining the primitive recursive categoricity spectrum of a structure as above, we require that each PR-degree $\mathbf{d} \in \mathtt{PRCatSpec}(A)$ primitively recursively computes both an isomorphism and its inverse.

In computability theory, the Turing degrees $\mathbf{0}, \mathbf{0}', \mathbf{0}'', \ldots$ serves as a 'spine' to classify and rank the algorithmic content of various objects or processes. In a similar vein, we define the following collections of PR-degrees.

DEFINITION 1.3 [3]. For each computable ordinal $\alpha > 0$, $\mathtt{Cone}(\Delta^0_\alpha)$ is the set containing all PR-degrees $\mathbf{d}$ such that $\mathbf{d} \geq_{PR} f$ for any total Δ^0_α-function f.

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

From Definitions 1.2 and 1.3, it is not difficult to obtain the following.

FACT 1.4. *If* A *is* Δ^0_α*-categorical, then* $\mathtt{Cone}(\Delta^0_\alpha) \subseteq \mathtt{PRCatSpec}(A)$.

In some sense, this is the analogue of Δ^0_α-categoricity of computable structures for punctual structures. In order to show that this bound is sharp, i.e., that Δ^0_α is the least primitive recursive degree of categoricity for A, we show that $\mathtt{PRCatSpec}(A) \subseteq \mathtt{Cone}(\Delta^0_\alpha)$, and hence $\mathtt{PRCatSpec}(A) = \mathtt{Cone}(\Delta^0_\alpha)$. This means that the PR-degrees which primitively recursively compute isomorphisms of A exactly coincide with the PR-degrees which primitively recursively compute Δ^0_α-functions, and thus, is analogous to the notion of degree of categoricity for computable structures.

Finally, we say that a PR-degree $\mathbf{d}$ is *low for punctual isomorphism* if for any structure $A \cong B$ such that $f : A \to B$ is an isomorphism and $f \oplus f^{-1} \leq_{PR} \mathbf{d}$, then A and B are punctually isomorphic. We say that a PR-degree $\mathbf{d}$ is a *degree of punctual categoricity* if there exists a punctual structure A such that $\mathbf{d}$ is the least degree in the spectrum $\mathtt{PRCatSpec}(A)$. To our best knowledge, the only known example of degrees of punctual categoricity is provided by the following result: if the graph of a total function $f(x)$ is primitive recursive, then $\deg(f)$ is a degree of punctual categoricity (Proposition 2.7 in [28]). We also cite [25, 27] for related results.

In this article, we study the primitive recursive categoricity spectra of injection structures, showing that they generally coincide with the analogous notions of Δ^0_n-categoricity. Nonetheless, in Section 3, we give an example of a computably categorical injection structure A for which $\mathtt{PRCatSpec}(A) \not\subseteq \mathtt{Cone}(\Delta^0_1)$, contrasting with the results obtained in Section 2 and the companion paper [4]. In the final section of this article, we also show that within each non-zero c.e. Turing degree, there are PR-degrees which exhibit 'contrasting' behaviour; in every non-zero c.e. Turing degree, there is a PR-degree low for punctual isomorphism and also a PR-degree that is a degree of punctual categoricity.

§2. Injection structures. An injection structure is a structure with a single unary function symbol, such that the function is injective. It is evident that such structures may be characterised by the number and types of orbits it contains. More specifically, each orbit is either finite, or is an ω-chain, or is a ζ-chain. We first show that such structures also possess punctual presentations.

THEOREM 2.1 (Folklore). *Let* M *be a computable injection structure with at least one infinite orbit. Then* M *has a punctual presentation.*

PROOF. Let M be a computable injection structure with at least one infinite orbit. Non-uniformly fix $N_0, N_1 \in \omega \cup \{\infty\}$ as the number of orbits in A of type ω and ζ, respectively. A punctual presentation of P of A may be obtained as follows. At each stage s, enumerate $n_0 + n_1$ many elements into P, where $n_i = \min\{s, N_i\}$, extending the infinite orbits in the obvious way. Since at least one of N_0, N_1 is non-zero, this ensures that P remains punctual. For any orbits of finite size, we simply wait for their size to be revealed in M (this is a c.e. wait), once revealed, we enumerate a finite cycle of the same size into P. ⊣

Recall that every computable injection structure is relatively Δ^0_3-categorical, and that the relatively Δ^0_2-categorical and relatively Δ^0_1-categorical injection structures

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

have also been characterised [10]. In the remainder of this section, we will study $\mathtt{PRCatSpec}(M)$, for all punctual injection structures M with at least one infinite orbit. Recall that a computable injection structure M is relatively Δ^0_1-categorical if and only if A is Δ^0_1-categorical if and only if M has only finitely many infinite orbits [10, Corollary 2.5].

THEOREM 2.2. *If* M *is a relatively* Δ^0_1*-categorical injection structure with at least one infinite orbit and infinitely many finite orbits, then* $\mathtt{PRCatSpec}(M) = \mathtt{Cone}(\Delta^0_1)$.

PROOF. From Fact 1.4, we have that $\mathtt{Cone}(\Delta^0_1) \subseteq \mathtt{PRCatSpec}(M)$. In order to prove the remaining inclusion, for any total computable g, we produce punctual injection structures A and B, both isomorphic to M, such that any isomorphism $h : A \to B$ is such that $g \leq_{pr} h \oplus h^{-1}$.

Encoding $g(0)$: Since g is a total computable function, all we need is to encode is the stage s at which $g(0)[s]\downarrow$. We shall do this by enumerating finite orbits into one structure A 'quickly' while 'delaying' their enumeration into B until after stage s is reached. The main tension here is that we have to keep both A and B punctual; we cannot arbitrarily delay enumerating elements. Since we assumed that M has at least one infinite orbit, and only finitely many infinite orbits, we can keep A, B punctual by simply enumerating these infinite orbits in both A, B. Moreover, to ensure that $A, B \cong M$, we never enumerate finite orbits into either A or B unless it has already been enumerated in M. As discussed, whenever a finite orbit, say of size n, is enumerated into M, we immediately enumerate a finite orbit of size n into A. Then once $g(0)\downarrow$, we also enumerate a finite orbit of size n into B. If $h : A \to B$ is an isomorphism, then by computing h on the elements within the finite orbit, we may recover the stage, where $g(0)$ converges.

Encoding $g(x)$*:* The obvious generalisation of the strategy described above would be to compute h on the first x many finite orbits revealed in A, and then arguing that at least one of these images must have index larger than the stage at which $g(x)$ converges. However, we note here that this process is not necessarily primitive recursive in $h \oplus h^{-1}$ and x. In particular, an arbitrary punctual injection structure M could enumerate finite orbits only at extremely sparse stages, meaning that the index of the x-th finite orbit in M cannot be obtained in a primitive recursive way.

To solve this issue, in addition to encoding the stage at which $g(x)$ converges, we also encode the stage $G(x+1)$ at which M enumerates the $(x+1)$-st finite orbit. More specifically, we enumerate the x-th finite orbit into B only after the $(x+1)$-st finite orbit has been enumerated into A. (Recall that A immediately enumerates any finite orbit enumerated into M.)

Now, we may define Ψ^h as follows. Non-uniformly fix $G(0)$, the stage at which A reveals the first finite orbit. From this stage, we may recover primitive recursively the indices of the elements in this finite orbit and then compute h on them. The image must produce indices large enough such that A has enumerated the next finite orbit and $g(0)$ has converged. Inductively, we may suppose that $\Psi^h(x) = g(x) \oplus G(x+1)$. To define $\Psi^h(x+1)$, use $G(x+1)$ to recover the indices of the $(x+1)$-st finite orbit in A and compute h on all elements of the first $x+1$ many finite orbits in A. Since B never enumerates the $(x+1)$-st finite orbit until A enumerates the $(x+2)$-nd finite orbit and $g(x+1)$ converges, one of the h-images must produce an index $\geq G(x+2), g(x+1)$. That is to say, $h \geq_{pr} g \oplus G \geq_{pr} g$. ⊣

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

Observe that the existence of an infinite orbit in M allows us to 'waste time' arbitrarily and delay enumerating the finite orbits into B. The proof above can thus be easily modified to fit the cases where there is some mechanism to 'waste time'. For instance, the result should also hold if M is relatively Δ^0_1-categorical and has some finite size repeated infinitely often. Recall that the remaining case for Δ^0_1-categorical injection structures are those with no infinite orbits and only finitely many orbits of each finite size. There are certain special cases in which we can apply a modification of the proof above to obtain $\mathtt{PRCatSpec}(M) \subseteq \mathtt{Cone}(\Delta^0_1)$, but surprisingly, it does not hold in general (see Theorem 3.1 for details).

A computable injection structure M is relatively Δ^0_2-categorical if and only if M is Δ^0_2-categorical if and only if M has finitely many ζ chains or finitely many ω chains [10, Corollary 3.3].

THEOREM 2.3. *If* $\mathbf{M}$ *is a relatively* Δ^0_2*-categorical injection structure which is not relatively* Δ^0_1*-categorical, then* $\mathtt{PRCatSpec}(M) = \mathtt{Cone}(\Delta^0_2)$.

PROOF. If an injection structure M is relatively Δ^0_2-categorical, but not relatively Δ^0_1-categorical, then its infinite orbits must be one of the following types:

(i) Finitely many ζ chains and infinitely many ω chains.
(ii) Only finitely many ω chains and infinitely many ζ chains.

Applying Fact 1.4, we clearly have that $\mathtt{Cone}(\Delta^0_2) \subseteq \mathtt{PRCatSpec}(M)$. It remains to prove that $\mathtt{PRCatSpec}(M) \subseteq \mathtt{Cone}(\Delta^0_2)$. That is, given g, a total Δ^0_2 function, we define two punctual injection structures A and B, both isomorphic to M, such that $g \leq_{pr} h \oplus h^{-1}$ for any isomorphism $h : A \to B$.

Fix a primitive recursive approximation g^* such that $\lim_s g^*(x,s) = g(x)$ for all x, and fix some injection structure of Type (i) that has a punctual presentation. In A and B, we never enumerate any finite orbits until they are observed to close in the given punctual (or computable) injection structure M. More specifically, we only enumerate ω chains while waiting for finite orbits to be enumerated into the given punctual injection structure. Similarly, we may non-uniformly fix the finitely many ζ chains and enumerate them in a standard way during the construction. This ensures that A and B are of the same isomorphism type as the given punctual injection structure M. Obviously, we shall use the ω chains to encode the stabilising stages s_x; $g^*(x, s_x) = g^*(x,s)$ for all $s \geq s_x$.

Encoding s_x: As mentioned above, we construct punctual structures $A = (\omega, f_A)$ and $B = (\omega, f_B)$ such that any isomorphism from A to B encodes g. In A, all ω chains will be standard; this allows us to access the left-most element of each ω chain in a primitive recursive way. We denote these elements as $a_{0,0}, a_{1,0}, a_{2,0}, \ldots$. At each stage s, enumerate $a_{s,0}$ and for each $a_{i,j}$ currently in A, where $f_A(a_{i,j})$ has yet to be defined, enumerate $a_{i,j+1} := f_A(a_{i,j})$. This ensures that f_A is primitive recursive and A is punctual.

In B, we shall use the left-most element of each ω chain in B to encode the stabilising stage s of $g^*(x,s)$ for various x. At stage 0, enumerate the element b_0 and let $b^l_{0,0} = b^r_{0,0} = b_0$. The idea is that $b^l_{i,s}$ and $b^r_{i,s}$ will denote the current left-most and right-most elements of the i-th ω chain at stage s, respectively. At each stage s, recursively suppose that $b^l_{i,s-1}$ and $b^r_{i,s-1}$ have been defined for $i < s$. If $g^*(x,s) \neq g^*(x,s-1)$, then do the following:

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

(1) For each j where $x < j < s$, define $f_B(b^r_{j-1,s-1}) = b^l_{j,s-1}$.
(2) For each $j < x$, let $b^l_{j,s} = b^l_{j,s-1}$ and $b^r_{j,s} = b^r_{j,s-1}$.
(3) Let $b^l_{x,s} = b^l_{x,s-1}$ and $b^r_{x,s} = b^r_{s-1,s-1}$.
(4) Pick fresh indices $i_{x+1}, i_{x+2}, \dots, i_s$, and enumerate the elements $b_{i_{x+1}}, b_{i_{x+2}}, \dots, b_{i_s}$, defining $b^l_{j,s} = b^r_{j,s} = b_{i_j}$ for each $x < j \leq s$.

Intuitively, if $g^*(x,s) \neq g^*(x,s-1)$, then we 'glue' the j-th chain for $x < j < s$ to the x-th chain at stage s, while leaving the first x many chains the same. This ensures that any element in the subsequent chains will have indices $> s$. Finally, to keep f_B primitive recursive, for each element $b \in B$ enumerated at stage $s-1$, enumerate some new element b' and let $f_B(b) = b'$ if $f_B(b)$ is not yet defined. In addition, if b is such that $b = b^r_{i,s}$ for some i, re-define $b^r_{i,s}$ to be b'.

Verification: Recall that any finite cycle or any ζ chain will be enumerated by 'copying' the given punctual injection structure. Thus, to verify that the construction produces the correct isomorphism type, it suffices to show that the procedure above produces infinitely many ω chains and nothing else. First, observe that for any b enumerated by the procedure above, $f_B(b)$ exists; if b was enumerated at stage s, by stage $s+1$, $f_B(b)$ must have been defined. Thus, there is no right-most element produced by the procedure above. Next, to show that each chain has a left-most element, we argue that for each $i \in \omega$, there exists $s \in \omega$ such that for all $t \geq s$, $b^l_{i,s} = b^l_{i,t}$. It follows directly from the description that $b^l_{0,t} = b^l_{0,0}$ for all $t \geq 0$. For $i > 0$, consider the stage s^* such that $g^*(x,s^*) = g^*(x,t)$ for all $t \geq s^*$ and for all $x < i$. Since g is a total Δ^0_2 function, such a stage s^* must exist. Following the construction, $f^{-1}_B(b^l_{i,s})$ is only ever defined when $g^*(x,s) \neq g^*(x,s-1)$ for some $x < i$. In particular, for all stages $t \geq s^* + 1$, we have that $g^*(x,t) = g^*(x,t-1)$ and so, $b^l_{i,t} = b^l_{i,s^*+1}$.

Finally, let $h : A \to B$ be an isomorphism. Define Ψ^h, a primitive recursive scheme as follows. Given x, compute $g^*(x, \max\{h(a_{i,0}) \mid i \leq x+1\})$. By pigeonhole principle, at least one of $h(a_{i,0}) = b^l_{j,s}$ for some $j \geq x+1$ and for some $s \in \omega$. It then follows that $\max\{h(a_{i,0}) \mid i \leq x+1\}$ is at least as big as the index of the left-most element of the $(x+1)$-st ω chain in B, which encodes a stage large enough such that g^* has stabilised for all $y \leq x$. In particular, $g^*(x, \max\{h(a_{i,0}) \mid i \leq x+1\}) = g(x)$.

For injection structures of Type (ii), we adopt a similar strategy. The only difference is that we must produce infinitely many ζ chains and nothing else when encoding g. That is, in addition to the procedure defined above, at each stage s, for each b currently defined as $b^l_{i,s}$ for some i, enumerate a new element b', define $f_B(b') = b$, and re-define $b^l_{i,s} = b'$. In A, we take a similar approach and extend the chains to the left at each stage by adding $a_{i,j}$ for $j < 0$ and letting $f_A(a_{i,j}) = a_{i,j+1}$ as before.

Using the modified construction, it is not too difficult to see that instead of ω chains, we now produce infinitely many ζ chains and nothing else. Furthermore, for each x, the $(x+1)$-st ζ chain in B has the property that all elements within it have indices $\geq s$, where s is such that for all $t \geq s$ and for all $y \leq x$, $g^*(y,t) = g^*(y,s)$. For the same reasons as before, $g(x) = g^*(x, \max\{h(a_{i,0}) \mid i \leq x+1\})$, and thus $g \leq_{pr} h$. Theorem 2.3 is proved. ⊣

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

THEOREM 2.4. *If* M *is a relatively* Δ^0_3*-categorical injection structure which is not relatively* Δ^0_2*-categorical, then* $\mathtt{PRCatSpec}(M) = \mathtt{Cone}(\Delta^0_3)$.

PROOF. Let M be a relatively Δ^0_3-categorical, not- Δ^0_2-categorical injection structure with a punctual presentation. That is, M has infinitely many ω chains and infinitely many ζ chains. This fact will be used implicitly in our encoding of $t_{x,s}$ and s_x for various x, s (the details will be provided shortly). Once again, by Fact 1.4, we have that $\mathtt{Cone}(\Delta^0_3) \subseteq \mathtt{PRCatSpec}(M)$, and it remains to prove that $\mathtt{PRCatSpec}(M) \subseteq \mathtt{Cone}(\Delta^0_3)$. Just as before, we focus only on the ω chains and ζ chains, enumerating finite orbits into our structures $A = (\omega, f_A)$ and $B = (\omega, f_B)$ only when they are shown to close in the given punctual copy M. Let g be a Δ^0_3 function and let $g^*(x, s, t)$ be a primitive recursive approximation to g; $\lim_s \lim_t g^*(x, s, t) = g(x)$. The strategy to encode g will be split into two parts. The first is to encode for each pair (x, s), the stage $t_{x,s}$ such that $g^*(x, s, t) = g^*(x, s, t_{x,s})$ for all $t \geq t_{x,s}$. The second is to encode the stage s_x for which $\lim_t g^*(x, s_x, t) = g(x)$. Then using the isomorphisms $h : A \to B$ and $h^{-1} : B \to A$, we recover both s_x and t_{x,s_x}, respectively, and compute $g^*(x, s_x, t_{x,s_x})$.

Encoding $t_{x,s}$: In A, for each (x, s), we keep a standard ω chain with left-most element $a_{\langle x,s\rangle,0}$. The idea is to encode $t_{x,s}$ using $h(a_{\langle x,s\rangle,0})$. Intuitively, this means that we should keep the indices of the left-most elements of each ω chain in B 'large enough'. In B, we begin with ω chains with left-most elements $b_{0,0}, b_{1,0}, b_{2,0}, \ldots$, respectively. At each stage t of the construction, we track the left-most elements of each of these ω chains in B, denoted by $b^l_{i,t}$. For each i, let $b^l_{i,0} = b_{i,0}$. At stage t, if $g^*(x, s, t) \neq g^*(x, s, t-1)$, then for each $i \geq \langle x, s\rangle$, enumerate some element b^* with some index $\geq t$ and define $f_B(b^*) = b^l_{i,t-1}$ and let $b^l_{i,t} = b^*$. Since $\lim_t g^*(x, s, t)$ exists for each (x, s), there must be some stage t^* for which $\lim_t g^*(x', s', t) = g^*(x', s', t^*)$ for each $x' \leq x$ and $s' \leq s$. After such a stage t^*, $b^l_{i,t}$ for each $i \leq \langle x, s\rangle$ never again changes. Furthermore, $b^l_{\langle x,s\rangle,t^*}$ must have index $\geq t^*$.

Encoding s_x: Let $P(x, s)$ be the predicate " $\lim_t g^*(x, s, t) = g(x)$." Since this is equivalent to "for all $s' \geq s$, $\lim_t g^*(x, s, t) = \lim_t g^*(x, s', t)$," $P(x, s)$ is a Π^0_2 predicate. We may further assume that there is a unique s for which $P(x, s)$ holds by requiring $s \geq x$ to be the least, and that for any $x < y$, if $P(x, s_x)$ and $P(y, s_y)$ are both true, then $s_x < s_y$. To keep the strategies disjoint, we encode the various s_x using the infinitely many ζ chains. In B, we enumerate standard copies of ζ chains, indexed by $c_{i,j}$ for $i, j \in \omega$, defining $f_B(c_{i,1}) = c_{i,0}$, $f_B(c_{i,2j}) = c_{i,2(j+1)}$ and $f_B(c_{i,2j+3}) = c_{i,2j+1}$. Within A, let the possible ζ chains be encoded by $d_{i,j}$ for $i, j \in \omega$. By a careful definition of f_A, for a fixed s, the collection of $d_{s,j}$ becomes a ζ chain iff $P(0, s) \vee P(1, s) \vee \cdots \vee P(s, s)$ holds. Intuitively, we 'grow' the chain of $d_{s,j}$ to the left only at stages where one of $P(x, s)$ currently holds for some $x \leq s$. Since $P(x, s)$ is Π^0_2, if it holds, then it must look to be true at infinitely many stages. Thus, at infinitely many stages, we extend the chain to the left, resulting in a ζ chain. Conversely, if there are infinitely many stages where we extend the chain containing $d_{s,0}$ to the left, at least one of $P(0, s), P(1, s), \ldots, P(s, s)$ must look to hold at infinitely many stages. That is to say, one of $P(x, s)$ is true for $x \leq s$. To recover s_x, we compute $h^{-1}(c_{i,0})$ for sufficiently many i, and we know that such an image must map to some $d_{s,j}$ contained in a ζ chain. Using pigeonhole principle, we then conclude that one such $d_{s,j}$ must have $s \geq s_x$.

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

Construction: We summarise the various types of elements in A and B below.

Element type	Purpose
$a_{i,j} \in A$, for all $i, j \in \omega$.	Standard ω chains; encoding $t_{x,s}$ where $i = \langle x, s\rangle$.
$b_{i,j} \in B$, for all $i \in \omega$ and all $j \geq i$.	Non-standard ω chains to be images for $h(a_{i,0})$.
$c_{i,j} \in B$, for all $i, j \in \omega$.	Standard ζ chains; encoding s_x where $i = \langle x, s\rangle$.
$d_{i,j} \in A$, for all $i, j \in \omega$.	ζ chain iff there exists $x \leq i$ such that $P(x, i)$ holds.

At stage 0, enumerate $a_{0,0}, d_{0,0}$ into A, $b_{0,0}, c_{0,0}$ into B. Define $b^l_{0,0} = b^r_{0,0} = b_{0,0}$ and $d^l_{0,0} = d^r_{0,0} = d_{0,0}$. At stage s, suppose that we have already defined for each $i < s$, $b^l_{i,s-1}, b^r_{i,s-1}, d^l_{i,s-1}$, and $d^r_{i,s-1}$.

Step 1: Enumerate $a_{s,j}$ for each $j < s$ and $a_{i,s}$ for each $i \leq s$ into A. Then define $f_A(a_{s,j}) = a_{s,j+1}$ for each $j < s$ and define $f_A(a_{i,s-1}) = a_{i,s}$ for each $i \leq s$. This ensures that at each stage s, $a_{i,j}$ for all $i, j \leq s$ has been enumerated and that f_A is primitive recursive on all such elements.

Step 2: Enumerate $b_{s,s}$ into B and define $b^l_{s,s} = b^r_{s,s} = b_{s,s}$. For each $i < s$, let j_i be the least index such that $j_i > j$ for any $b_{i,j}$ currently enumerated into B. If there is some $\langle x, n\rangle = i < s$ and $g^*(x, n, s) \neq g^*(x, n, s-1)$, then enumerate b_{k,j_k} and b_{k,j_k+1} into B for all k such that $i \leq k < s$, defining

- $b^r_{k,s} = f_B(b^r_{k,s-1}) = b_{k,j_k+1}$,
- $b^l_{k,s} = b_{k,j_k}$, and
- $f_B(b_{k,j_k}) = b^l_{k,s-1}$.

Otherwise, only enumerate b_{i,j_i} into B, define $b^r_{i,s} = f_B(b^r_{i,s-1}) = b_{i,j_i}$ and $b^l_{i,s} = b^l_{i,s-1}$.

Step 3: Enumerate $c_{s,2j}, c_{s,2j+1}$ for each $j < s$ and $c_{i,2s}, c_{i,2s+1}$ for each $i \leq s$ into B. Now define $f_B(c_{i,2(s-1)}) = c_{i,2s}$, and $f_B(c_{i,2s+1}) = c_{i,2s-1}$ for each $i < s$, and

$$c_{s,2s+1} \xrightarrow{f_B} c_{s,2s-1} \xrightarrow{f_B} \ldots \xrightarrow{f_B} c_{s,1} \xrightarrow{f_B} c_{s,0} \xrightarrow{f_B} c_{s,2} \xrightarrow{f_B} \ldots \xrightarrow{f_B} c_{s,2s}.$$

At each stage s, we have that B contains $c_{i,j}$ for each $i \leq s$ and each $j < 2(s+1)$, and f_B is primitive recursive on all such elements.

Step 4: Enumerate $d_{s,0}$ into A, and define $d^l_{s,s} = d^r_{s,s} = d_{s,0}$. For each $i < s$, let j_i be the least index such that $j_i > j$ for any $d_{i,j}$ currently enumerated into A. If there is some $x \leq i < s$ such that $P(x, i)$ *fires* (looks to be true at the current stage), then enumerate d_{i,j_i} and d_{i,j_i+1} into A, and define the following:

- $d^r_{i,s} = f_A(d^r_{i,s-1}) = d_{i,j_i+1}$,
- $d^l_{i,s} = d_{i,j_i}$, and
- $f_A(d_{i,j_i}) = d^l_{i,s-1}$.

Otherwise, only enumerate d_{i,j_i} into A, and define $d^r_{i,s} = f_A(d^r_{i,s-1}) = d_{i,j_i}$. Also let $d^l_{i,s} = d^l_{i,s-1}$.

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

Whenever a finite orbit in the given punctual injection structure is revealed, we also enumerate a finite orbit of the same size into the structures A and B.

Verification: We make the following observations:

(i) For any x, s, the collection $\{b_{\langle x,s\rangle,j} \mid j \geq \langle x,s\rangle\}$ forms an ω chain with left-most element $b_{\langle x,s\rangle,t}$, where for all $t' \geq t$ and for any $\langle x',s'\rangle \leq \langle x,s\rangle$, $g^*(x',s',t') = g^*(x',s',t)$. (Note that $b_{\langle x,s\rangle,j}$ for $j < \langle x,s\rangle$ is undefined as they are never enumerated into B.)

For a fixed pair x, s, since $\lim_t g^*(x,s,t)$ is assumed to exist, then there must be some finite stage $t_{x,s}$ such that for all $t \geq t_{x,s}$, $g^*(x,s,t_{x,s}) = g^*(x,s,t)$. This implies that for all $t > \max\{t_{x',s'} \mid \langle x',s'\rangle \leq \langle x,s\rangle\}$, we always define $b^l_{\langle x,s\rangle,t+1} = b^l_{\langle x,s\rangle,t}$ and $f_B^{-1}(b^l_{\langle x,s\rangle,t}) \uparrow$. Thus, there is some finite stage t after which the left-most element in the collection $\{b_{\langle x,s\rangle,j} \mid j \geq \langle x,s\rangle\}$ never again changes. That is, the collection forms an ω chain. From the argument above and an analysis of the actions in Step 2, it is easy to conclude that the index $j \in \omega$ for which $b_{\langle x,s\rangle,j} = b^l_{\langle x,s\rangle,t}$ is such that $j \geq t_{x',s'}$ for any $\langle x',s'\rangle \leq \langle x,s\rangle$. Therefore, (i) is true.

(ii) For any $s \in \omega$, the collection $\{d_{s,j} \mid j \in \omega\}$ forms a ζ chain iff there exists some $x \leq s$ such that $P(x,s)$ fires at infinitely many stages.

To see why (ii) holds, we turn our attention to Step 4 of the construction. First suppose that for all $x \leq s$, $P(x,s)$ fires only finitely often. In particular, there is some finite stage s^* large enough such that after s^*, $P(x,s)$ never again fires for all $x \leq s$. An analysis of Step 4 in the construction will allow one to conclude that for all $s' \geq s^*$, $d^l_{s,s'} = d^l_{s,s^*}$, and that $f_A^{-1}(d^l_{s,s^*}) \uparrow$. In other words, the collection $\{d_{s,j} \mid j \geq s\}$ forms an ω chain.

On the other hand, if there is some $x \leq s$ for which $P(x,s)$ fires infinitely often, then let these stages be denoted by $s_0 < s_1 < s_2 < \ldots$. Obviously, at all such stages, we would have defined $d^l_{s,s_i} = f_A^{-1}(d^l_{s,s_i-1})$, meaning that the collection $\{d_{s,j} \mid j \in \omega\}$ has no left-most element. Furthermore, as described in Step 4, we always define $d^r_{s,s'} = f_A(d^r_{s,s'-1})$, thus also ensuring that $\{d_{s,j} \mid j \in \omega\}$ has no right-most element. Thus, $\{d_{s,j} \mid j \in \omega\}$ forms a ζ chain iff there is some $x \leq s$ for which $P(x,s)$ fires infinitely often.

To see that A and B are of the correct isomorphism type, it suffices to show that they consist of infinitely many ω chains and infinitely many ζ chains, as the finite orbits are enumerated whenever they are observed to close in the given punctual copy. Step 3 of the construction ensures that B consists of ζ chains of the form $\{c_{i,j} \mid j \in \omega\}$ for each $i \in \omega$. By applying (i), we may also conclude that B contains infinitely many ω chains given by $\{b_{i,j} \mid j \geq i\}$ for each $i \in \omega$. In fact, it is not difficult to see that these are exactly all the ω chains in B.

In A, it is easy to see that there are infinitely many ω chains, given by $\{a_{i,j} \mid j \in \omega\}$ for each $i \in \omega$. Since $g(x)$ is a total Δ^0_3 function, for each $x \in \omega$, there must be some $s_x \in \omega$ such that $P(x,s_x)$ fires infinitely often. Applying (ii) from above, we may then obtain that the chains $\{d_{s_x,j} \mid j \in \omega\}$ for each $x \in \omega$ are all ζ chains. Thus, A contains both infinitely many ω chains and also infinitely many ζ chains. We also note here that as a consequence of (ii), the only ζ chains in A are of the form $\{d_{s_x,j} \mid j \in \omega\}$ for some $x \in \omega$.

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

Let $h : A \to B$ be an isomorphism. To recover $g(x)$, consider the following procedure. Compute $h^{-1}(c_{i,0})$ for each $i \leq x$. Since h is an isomorphism, we obtain that $h^{-1}(c_{i,0})$ must also be contained in some ζ chain of A. Recall that the only ζ chains of A are of the form $\{d_{s_y,j} \mid j \in \omega\}$ for some $y \in \omega$. By pigeonhole principle, at least one of $h^{-1}(c_{i,0}) = d_{s,j}$ for some j and s where $s \geq s_x$. Let s^* be the maximum of all such s obtained from $h^{-1}(c_{i,0})$. Clearly, s^* has the property that $\lim_t g^*(x, s^*, t) = \lim_t g^*(x, s', t)$ for any $s' \geq s^*$. Or equivalently, $\lim_t g^*(x, s^*, t) = \lim_s \lim_t g^*(x, s, t)$.

Given s^* from above, compute $h(a_{i,0})$ for all $i \leq \langle x, s^* \rangle$. Each of these images must be the left-most element of an ω chain in B. Since B only contains ω chains of the form $\{b_{\langle y,s\rangle,j} \mid j \geq \langle y, s\rangle\}$, we have that each $h(a_{i,0}) = b_{\langle y,s\rangle,t}$ for some $y, s, t \in \omega$. Pick t^* with the following properties:

- $h(a_{i^*,0}) = b_{\langle y,s\rangle,t^*}$ for some $i^* \leq \langle x, s^* \rangle$, and
- for any other $b_{\langle y',s'\rangle,t'} = h(a_{i,0})$, $\langle y', s' \rangle \leq \langle y, s \rangle$.

By pigeonhole principle, it must be that $\langle x, s^* \rangle \leq \langle y, s \rangle$, and by (i), t^* has the property that for any $t \geq t^*$, $g^*(x, s^*, t^*) = g^*(x, s^*, t)$. In conclusion, $g^*(x, s^*, t^*) = \lim_t g^*(x, s^*, t) = \lim_s \lim_t g^*(x, s, t)$. Since the procedure described above is clearly primitive recursive, $g \leq_{pr} h \oplus h^{-1}$, and thus, $\mathtt{PRCatSpec}(M) \subseteq \mathtt{Cone}(\Delta^0_3)$. Theorem 2.4 is proved. ⊣

§3. A pathological injection structure.

THEOREM 3.1. *There exists a structure* A, *computably categorical but not punctually categorical, such that* $\mathtt{PRCatSpec}(A) \not\subseteq \mathtt{Cone}(\Delta^0_1)$.

We shall construct our structure to be a punctual injection structure. Recall from Section 2 that this injection structure necessarily has no infinite orbits, and no orbit sizes repeated infinitely often. The intuitive idea for the proof would be to *press* the 'opponents' into copying our structure as much as possible, i.e., ensuring that either their structure is d-punctually isomorphic to ours, (for some constructed function oracle d), or not isomorphic at all.

PROOF. Fix some computable but not primitive recursive function g, let $\{\Psi_e\}_{e\in\omega}$ be a list of all primitive recursive schemes, and let $\{B_e\}_{e\in\omega}$ be a list of all punctual structures in the language of one unary function symbol. We construct d, a Δ^0_2 function oracle, and a punctual injection structure $A = (\omega, f_A)$, satisfying the following requirements:

$P_e : \Psi^d_e \neq g$.

$Q_e : d \geq_{pr} q_e \oplus q_e^{-1}$ where $q_e : A \to B_e$ is an isomorphism, or $A \not\cong B_e$.

Strategy for one Q_e requirement: Fix some punctual structure B_e with a single unary function. If such a function is ever discovered to not be injective, then Q_e is trivially satisfied. We may thus assume that B_e is an injection structure.

When the construction begins, we enumerate only finite orbits of some given size n. Although B_e is punctual and should enumerate its elements 'quickly', we note that arbitrary B_e could still be much 'slower' than the structure A defined by us. The intention here is obviously to use d to encode this delay. More specifically, since

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

we only enumerate finite orbits of size n, if B_e ever shows some orbit not of size n (or more generally if B_e is not a substructure of A), then we will never enumerate such a size into our structure A, ensuring that $A \not\cong B_e$. Thus, B_e should eventually enumerate an orbit of size exactly n. Once it does so, we define $d(0)$ to be the stage number at which this happens. The intention here is that $d(j)$ encodes the stage number at which B_e enumerates the j-th orbit enumerated into A.

Strategy for one Q_e and infinitely many P_e: Fix some infinite sequence of sizes $n_0 < n_1 < n_2, \ldots$, and begin the strategy for Q_e as described previously. Enumerate only orbits of size n_0 and define $d(j)$ to be the stages at which B_e enumerates a suitable q_e-image for the j-th orbit in A. Repeat this process until $\Psi_0^d \neq g$ is witnessed. Since g is not primitive recursive and d currently contains only primitive recursive information, $\Psi_0^d \neq g$ must be witnessed at some finite stage s_0. Once such a stage is reached, we define $d(s_0)[s_0] = 0$. The idea is that $d(s_0)$ should encode a stage large enough such that B_e contains $A[s_0]$ as a substructure. As long as $A[s_0]$ is not a substructure of B_e, $d(s_0)$ remains at 0.

Suppose that for all $j < i$, we have defined s_j, and $\Psi_j^d \neq g$ is witnessed by the initial segment $d \upharpoonright s_j$. We now begin enumerating only orbits of size n_i into our structure A. Just as before, if it is discovered that B_e is not a substructure of A, then Q_e will be satisfied by keeping $A \not\cong B_e$. Similarly, $d(s_{i-1} + k)$ will encode the stage at which the k-th orbit of size n_i is enumerated into B_e. Continue this process until it is discovered that $\Psi_i^d \neq g$. Since d consists of only primitive recursive information (with finitely many exceptions, i.e., $d(s_j)$ for each $j < i$) and g is not primitive recursive, $\Psi_i^d \neq g$ must be witnessed at some finite stage s_i. We may then recursively repeat this procedure as described above, defining $d(s_i)[s_i] = 0$ and enumerating orbits of size n_{i+1} into A.

However, for $i > 0$, the failure of Ψ_i^d to compute g is witnessed by the initial segment $d \upharpoonright s_i$. In particular, this initial segment includes $d(s_j)$ for $j < i$. At stages s where $A[s_j]$ is discovered to be a substructure of B_e, the value of $d(s_j)$ changes from 0 to s, possibly changing the computation in such a way that Ψ_i^d now looks to be equal to g. When this happens, we initialise all requirements P_k for $k > j$ and return to enumerating only orbits of size n_{j+1} and restarting our definition of s_k for all $k > j$. Clearly, each P_i can only be initialised if $d(s_j)$ changes for some $j < i$, and thus each P_i will be initialised only finitely often.

The general strategy: We order the requirements $Q_0, P_0, Q_1, P_1, \ldots$. During the construction, we maintain $n_0 < n_1 < \ldots$ and $s_0 < s_1 < \ldots$ as distinct sets of *markers* with the usual rules; whenever n_i or s_i changes, all n_j or s_j for $j > i$ will be *kicked* to some value $> n_i$ or $> s_i$, respectively. As described earlier, we begin by enumerating orbits of size n_0 into A and record the stages at which these orbits are enumerated into B_0. At each stage s of the construction, as long as $\Psi_0^{d\upharpoonright s} = g$, we kick s_i for all $i \geq 0$ and define $s_0 = s + 1$. Once $\Psi_0^{d\upharpoonright s} \neq g$ is witnessed, let $s_0 = s$ and define $d(s_0) = 0$.

Suppose now that we have currently witnessed $\Psi_j^{d\upharpoonright s_j} \neq g$ for each $j < i$. We will attend to $Q_0, Q_1, \ldots, Q_i$ and P_i in subsequent stages. Similar to before, we enumerate only orbits of size n_i while waiting for the stage s such that $\Psi_i^{d\upharpoonright s} \neq g$. As long as $\Psi_i^{d\upharpoonright s} = g$, we kick s_k for all $k \geq i$ at stage s. For inputs $x > s_{i-1}$, we use $d(x)$ to record the stage at which the $(x - s_{i-1})$-th orbit of size n_i is enumerated into the

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

structures $B_0, B_1, \dots, B_i$. More specifically, we define $d(s_{i-1}+k)$ to be an $(i+1)$-tuple, where each bit is the stage at which B_j enumerates the k-th (>0) orbit of size n_i for each $j \leq i$. We continue this process until the first stage s where one of the following happens:

- If $\Psi_i^{d\restriction s} \neq g$, then define $d(s_i)[s]$ to be the all zero i-tuple, and proceed to attend to $Q_0, Q_1, \dots, Q_{i+1}$, and begin the process to find s_{i+1}.
- For some $j < i$, $A[s_j]$ is discovered to be a substructure of B_k for some $k \leq j$ at stage s. At such a stage s, change the k-th bit of $d(s_j)$ to s (recall that $d(s_j)$ is a j-tuple), define $s_{j+1} = s+1$ and initialise the requirements $P_{j'}, Q_{j'}$ for all $j' > j$. Then return to attending to $Q_0, Q_1, \dots, Q_{j+1}$ and restart the process to find s_{j+1}.

Intuitively, the first case is when we manage to successfully define s_i without any changes to $d \upharpoonright s_{i-1}$. The latter case is where some change is made in $d \upharpoonright s_{i-1}$ before finding s_i. When this happens, we return to the strategy for attending to the Q-requirements with indices $\leq j+1$, and P_{j+1} for the largest j where $d \upharpoonright s_j$ did not change. We then also restart the definition of $d(x)$ for $x > s_j$. Note that this does not injure the strategy for $Q_0, \dots, Q_j$ at all; we only need ensure that d computes an isomorphism between A and B_k for $k \leq j$. Therefore, only the strategies for the requirements $P_{j'}$ for $j' > j$ are 'injured' by this action.

There is an additional point to note when we have multiple Q_e requirements. Recall that for a single Q_e, if for some k, B_e reveals more orbits of size k than A currently contains, then we are able to permanently keep $A \not\cong B_e$. However, the different Q_e might possibly interact in a way that 'damages' our strategy of recovering an isomorphism from d. In particular, the information we have encoded into d is not quite sufficient; we need to be able to compute where each structure B_e places the orbits of size k for $n_j < k < n_{j+1}$ (which might have been enumerated into A before k is restrained from A). Thus, we shall also track the stages t_e at which we 'act' for Q_e (see the construction for the definition). If we never do so, then let $t_e \uparrow$. Otherwise, $d(s_j)$ will also encode the stage at which $A[t_{j+1}]$ becomes a substructure of B_l for various $l \leq j$. Since $A[t_{j+1}]$ being a substructure of B_l is c.e., it is easy to see that the 'injury' to the P-requirements remain finite. (Recall that Q-requirements are never injured.)

Construction: Each stage of the construction could possibly last for a non-primitive recursive amount of time. During the construction, we also track sub-stages; stages of primitive recursive time at which we enumerate new elements into A.

At stage 0, define $n_i[0] = i+1$ and $s_i = i$ for each $i \in \omega$. Begin enumerating orbits of size $n_0[0]$ into A, and do the following:

(i) If B_0 remains a substructure of A, then within primitive recursive time, B_0 must enumerate the first orbit of size $n_0[0]$. When it does so, we record the sub-stage at which this happens using $d(0)$.

(ii) If B_0 is not a substructure of A, then define $d(0) = 0$. Since n_0 is currently 1, and A only contains orbits of size $n_0[0]$, in order for B_0 to not be a substructure of A, one of the following must happen:
- B_0 enumerates an orbit which has yet to close. By ensuring that our structure has no infinite orbits, B_0 can only be isomorphic to A if all orbits of B_0 are

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

also finite. Thus, any orbit enumerated by B_0 has to eventually close. That is, we need not consider orbits which do not close.

- B_0 enumerates a (closed) orbit of size $k > n_0[0]$. Then define $n_0[1]$ to be some number $> k$, kicking all subsequent n_j for $j > 0$. Throughout the construction, we shall only enumerate orbits of size $n_i[s]$ for some i at each stage s of the construction. This ensures that we shall never enumerate any orbit of size k into A, as $k < n_0[1] \leq n_0[s] < n_1[s] < \ldots$ for all $s > 0$.
- B_0 enumerates more orbits of size n_0 than A. Then we similarly define $n_0[1]$ to be a fresh number. In subsequent stages of the construction, we will never enumerate any orbit of size $n_0[0]$.

If n_0 is defined to be some fresh number by an action above, then we say to have *acted* for Q_0, and define t_0 to be the sub-stage at which we did so.

Once we have succeeded in defining $d(0)$, proceed to stage 1.

At stage s, we say that

- Q_i *requires attention* if we have never *acted* for Q_i (to be defined later), and there is some orbit of size $k \geq n_i[s]$, such that B_i contains more orbits of size k than A, and
- P_i *requires attention* if $\Psi_i^{d\restriction s}$ is currently equal to g.

In addition, if we have acted for Q_i, then let t_i be the sub-stage at which we did so. At the beginning of stage s, for any $i \geq 0$ and $j \leq i$ such that $s_i < s$ and we have yet to act for Q_j, do the following:

- If t_{i+1} is defined, and the structures $A[s_i]$ and $A[t_{i+1}]$ are both substructures of B_j, then define the j-th bit of $d(s_i)$ to be the sub-stage at which this was first witnessed. Otherwise, let the j-th bit of $d(x)$ for all $x > s_{i-1}$ be 0.
- If t_{i+1} is undefined, then we only check whether $A[s_i]$ is a substructure of B_j. If it is, then define the j-th bit of $d(s_i)$ to be the first sub-stage at which it was witnessed. Otherwise, define the j-th bit of $d(x)$ for all $x > s_{i-1}$ to be 0.

Once all such changes are made, let i be the least for which Q_i or P_i requires attention.

If it was Q_i which required attention, then define $n_i[s+1]$ to be some fresh number, and proceed to stage $s+1$. We then say that we have *acted* for Q_i and define t_i to be the current sub-stage.

If it was P_i which required attention, then define $s_i = s+1$ and define $d(x)$ for all $s_{i-1} < x < s_i = s+1$ as follows. While attempting to define $d(x)$, we enumerate only orbits of size $n_i[s]$ into A. Let $j \leq i$ be given.

(i) Define the j-th bit of $d(x)$ to be 0 if one of the following holds:
- B_j is currently not a substructure of A. To be more specific, this means that there is some finite size k such that B_j currently has more orbits of size k than A does.
- The j-th bit of $d(y)$ is currently 0 for some $y < x$. Intuitively, this means that B_j currently does not look isomorphic to A.
- We have acted for Q_j at some earlier stage. In fact, once we have acted for Q_j, let the j-th bit of $d(x)$ for all $x > s_{j-1}$ be 0.

In the following cases, we may thus assume that none of the above hold.

(ii) Define the j-th bit of $d(x)$ to be the sub-stage at which the $(x - s_{i-1})$-th orbit of size $n_i[s]$ is enumerated into B_j. Since B_j is punctual, it has to enumerate

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

new elements primitive recursively. In addition, new orbits either grow in size quickly or close and reveal their final size. Applying this with the fact that there are only a fixed number of orbits of size $< n_i[s]$ currently in A, and we are only enumerating orbits of size $n_i[s]$ into A, after some primitive recursive delay, one of the following must happen:

- B_j enumerates the $(x - s_{i-1})$-th orbit of size $n_i[s]$. As mentioned earlier, we record the sub-stage at which this happens in the j-th bit of $d(x)$.
- There is some $k \geq n_j[s]$ such that B_j has more orbits of size k than A does. That is, Q_j would require attention at the next stage, and if acted for, we claim that we never again enumerate any orbits of size k (to be verified later).
- There is some $k < n_j[s]$ such that B_j has more orbits of size k than A does. In this case, we cannot forcefully kick higher priority $n_{j'}[s] \leq k$ for the sake of Q_j. We instead ensure that each size appears only finitely often in A, meaning that we can non-uniformly fix $q_j^{-1} : B_j \to A$ on all orbits of size $< n_j$, allowing us to satisfy Q_j via computing $q_j \oplus q_j^{-1}$ from d (verified below). In the meantime, define the j-th bit of $d(y)$ to be 0 for each $y > s_{j-1}$.

Once we have successfully defined $d \upharpoonright (s+1)$, then proceed to stage $s+1$.

Verification: We first make some preliminary observations. Notice that each n_e only changes if $n_{e'}$ for some $e' < e$ changes, or we acted for Q_e during the construction. Since we only act at most once for each Q_e, it is easy to see that each n_e stabilises eventually.

To see that d is Δ^0_2, observe that for each $x \in \omega$, and for each s_i, exactly one of $x > s_i$, or $x \leq s_i$ is true for cofinitely many stages of the construction. Note that this is simply a consequence of how each s_i only increases when it is changed, and that we do not claim or require that s_i stabilises here. For a given x, since $x = s_x[0]$, there exists a least i for which $x \leq s_i$ for cofinitely many stages. As a result, $d(x)$ must be an $(i+1)$-tuple. For such an i, there is a stage s^* such that n_i never again changes after stage s^*. We may further assume inductively that s^* is also such that $d \upharpoonright x$ never changes after s^*. We argue that each bit of $d(x)$ stabilises. Fix some $j \leq i$ and consider the j-th bit of $d(x)$:

- If there is some $y < x$ such that the j-th bit of $d(y)$ is 0, then the j-th bit of $d(x)$ will also be defined to be 0. Furthermore, by the assumption that $d \upharpoonright x$ never again changes after stage s^*, the j-th bit of $d(x)$ will forever remain at 0.
- We may thus suppose that the j-th bit of $d(y)$ is always > 0 for each $y < x$. We further split into the cases where either $x < s_i$ or $x = s_i$. For $x < s_i$, recall that the j-th bit of $d(x)[s]$ either records the sub-stage at which B_j enumerates the $(x - s_{i-1})$-th orbit of size $n_i[s]$, or records 0 if B_j was discovered to not be a substructure of A at stage s. This means that there is some k such that $B_j[s]$ contains more orbits of size k than $A[s]$ does. There are then two further possibilities:
  - If $k \geq n_j[s]$, then we must have already acted for Q_j, otherwise $n_j[s+1] \neq n_j[s]$ a contradiction to the assumption that n_i, and thus n_j, has already stabilised. However, if we have already acted for Q_j, then the j-th bit of $d(y)$ for all $y > s_{j-1}$ is always defined to be 0. In particular, we obtain that

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

the j-th bit of $d(x)$ remains at 0 for the remainder of the construction as $x > s_{i-1} \geq s_{j-1}$ for any $j \leq i$.

– If $k < n_j[s]$, then recall from the construction (stage s (ii)) that the j-th bit of $d(x)$ is also defined to be 0. We claim that it never again changes after stage s^*. Suppose for a contradiction that it does. Then that implies that we must have enumerated new orbits of size k into A, allowing B_j to become a substructure of A. This means that k is necessarily $n_l[s]$ for some $l < j$. However, this further implies that at some stage $t > x$, we returned to attending to P_l; meaning that $s_l > x$, a contradiction to the assumption that $i > l$ is the least for which $s_i \geq x$ for cofinitely many stages.

For $x = s_i$, the j-th bit of $d(x)$ records one of the following:
– the sub-stage at which $A[s_{i-1}]$ becomes a substructure of B_j, or
– the sub-stage at which $A[t_i]$ becomes a substructure of B_j, provided t_i is defined and larger than s_{i-1}, or
– 0 otherwise.

It is easy to see that the first two points are c.e. events once s_{i-1} has stabilised and t_i has been defined if ever. Thus, $d(x)$ never changes after some finite stage.

In conclusion, we have that each n_e eventually stabilises, and that d is a Δ^0_2 function. We are now ready to prove that P_e and Q_e are satisfied.

Each P_e is satisfied: We proceed by induction on $e \in \omega$ and show that s_e eventually stabilises at some finite value. Let e be the least such that s_e changes infinitely often. From the construction, s_e only changes if either Q_e or P_e requires attention or some $s_{e'}$ where $e' < e$ changes. Since e is assumed to be the least for which s_e changes infinitely, then we may assume that either Q_e or P_e requires attention infinitely often:

- Recall that Q_e can only require attention at stage s if we have never acted for it. In particular, since s_e is the least which changes infinitely often, Q_e must eventually be the highest priority requirement which requires attention. At such a stage, we would have acted for Q_e, and it will never again require attention in the construction. We may thus suppose that it is P_e which requires attention infinitely often.
- Suppose that P_e requires attention infinitely often. Let s^* be a stage large enough such that all of the following hold:
  – $s_0, s_1, \ldots, s_{e-1}$ and $n_0, n_1, \ldots, n_e$ have all stabilised.
  – $d(x)$ for all $x \leq s_{e-1}$ does not change after stage s^*.
  – P_e requires attention at stage s^*.

  By our assumption that e is the least for which s_e changes infinitely and the earlier observations, such an s^* must exist.

  If there is some stage $s > s^*$ at which P_e does not require attention, then we claim that P_e never again requires attention. Let s denote the first stage after s^* at which P_e does not require attention. Similarly, let s' denote the first stage after s at which P_e again requires attention. Recall that P_e requires attention at some stage iff Ψ_e temporarily computes g with the current d. More specifically, we have that $\Psi_e^{(d \upharpoonright s)[s])} \neq g$, $\Psi_e^{(d \upharpoonright s')[s']} = g$, and at some stage s'' between s and s', $d(x)[s''] \neq d(x)[s''+1]$ for some $x < s$. It follows from the construction

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

that $s_e[s] = s$, and by the choice of s^*, we obtain that $s_{e-1} < x < s_e[s'']$. For each such x, $d(x)$ records either 0 in the j-th bit or the sub-stage at which B_j enumerates the $(x - s_{e-1})$-th orbit of size n_e. Consider the following cases:

– If the j-th bit of $d(x)[s]$ is non-zero, then it means that B_j has enumerated the $(x - s_{e-1})$-th orbit of size n_e. Since B_j is punctual, this enumeration cannot be 'undone', and the j-th bit of $d(x)$ cannot change unless the j-th bit of $d(y)$ changes to 0 for some $y < x$. By choice of s^* such a y must be larger than s_{e-1}, as $d \upharpoonright (s_{e-1} + 1)$ is assumed to have stabilised. But the j-th bit of $d(y)$ records the sub-stage at which B_j enumerates the $(y - s_{e-1})$-th orbit of size n_e. This means that the j-th bit of $d(y)$ cannot possibly be 0.

– As noted above, if there is some $y < x$ such that the j-th bit of $d(y)$ is 0, then the j-th bit of $d(x)$ will similarly be 0. We may thus assume that x is the least for which the j-th bit of $d(x)[s]$ is 0. An analysis of the actions in the construction allows one to easily conclude that this can only happen if there is some $k \in \omega$ such that $B_j[s]$ contains more orbits of size k than A. Obviously, $k < n_j \leq n_e$, otherwise, we would have acted for Q_j at some stage after s^*, contradicting the assumption that n_j has stabilised. Note that we could not have acted for Q_j before s^*, otherwise the j-th bit of $d(y)$ would be 0 for all $y \geq s^*$ (by definition of $d(x)$).

In other words, $d(x)[s]$ once defined, never changes again for each $x < s$, implying that $g \neq \Psi_e^{(d \upharpoonright s)[s]} = \Psi_e^{d \upharpoonright s}$. That is, P_e never again requires attention after stage s, contradicting the assumption that it requires attention infinitely often. Thus, we may assume that for all stages after s^*, P_e always requires attention.

In addition, recall that we may discover in primitive recursive time whether B_j enumerates the $(x - s_{e-1})$-th orbit of size n_e, or enumerates more orbits of size k than A does for some $k < n_e$. Thus, for each $x > s_{e-1}$, $d(x)$ can be computed primitive recursively. But this implies that $\Psi_e^d = g$, contradicting the assumption that g is not primitive recursive. Therefore, P_e cannot require attention for infinitely many stages.

It follows immediately that for each e, P_e is satisfied as $\Psi_e^{d \upharpoonright s_e} \neq g$, and thus $\Psi_e^d \neq g$.

Each Q_e is satisfied: During the construction, note that we enumerate orbits of size n_e only at stages where P_e is the highest priority requirement which requires attention. Using the observation that s_e also stabilises eventually, it follows that after stage s_e, we never enumerate any orbit of size $k \leq n_e$ into A. Therefore, A only consists of finitely many orbits of size k for each $k \in \omega$.

It remains to argue that each Q_e is satisfied. Since each n_e stabilises at a finite value and we only ever enumerate orbits of size $n_e[s]$ for some e, s, A contains no infinite orbits. We may thus assume that B_e also contains no infinite orbits, as B_e cannot possibly be isomorphic to A otherwise. We split the proof into two cases as follows.

First, suppose that at some finite stage s, we acted for Q_e. Recall that this means that at stage s, Q_e is the highest priority requirement which required attention. In particular, we have yet to act for Q_e at stage s, and there is some orbit of size $k \geq n_e[s]$ such that B_e contains more orbits of size k than A. We claim that after we act for Q_e, no more orbits of size k will be enumerated into A, thus preserving

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

$B_e \not\cong A$. When we act for Q_e, we define n_e to be some fresh number at stage s, and kick all subsequent $n_{e'}$ for $e' > e$. That is to say, $k < n_{e'}[s']$ for each $e' \geq e$ and $s' > s$. By the properties of the markers n_i, we also have that $n_i[s] < n_e[s] \leq k$ for each $i < e$, and thus $k \neq n_i[s]$. Therefore, if we never act for any Q_i for $i < e$ after stage s, B_e cannot possibly be isomorphic to A, as no more orbits of size k are ever enumerated. On the other hand, if there is some stage $s' > s$ during which we act for Q_i where $i < e$, n_i is similarly defined to be a fresh number at such a stage s'. This once again ensures that $n_i[s''] > k$ for any $i \in \omega$ and any $s'' > s'$. In conclusion, once we have acted for Q_e, we never again enumerate any orbit of size k into A.

Now suppose that we never act for Q_e during the construction. If Q_e requires attention at infinitely many stages, then there must be some finite stage during which we act for it. Thus, there must be some stage s after which Q_e never again requires attention. In particular, after such a stage, we must have that B_e is always a substructure of A; as long as we are able to define $q_e : A \to B_e$ on orbits enumerated into A after such a stage, q_e^{-1} is naturally also primitive recursive. We fix such a stage as s^*, and define $q_e : A \to B_e$ as follows. Let k be the largest such that there is an orbit of size k in B_e at stage s^*. In addition, let $i \geq e$ be the least such that $n_i > k$. Observe that after stage s_i in the construction, we never enumerate any orbit of size $< n_i$. In particular, by non-uniformly fixing q_e on $A \upharpoonright s_i$, we need only define q_e on orbits of size $\geq n_i > k$.

Let $a_0, a_1, \ldots$, be the orbits enumerated into A after stage s_i. We further assume that for each stabilising value of n_j, we enumerate at least one orbit of size n_j into A. In particular, for each $j \in \omega$, a_j has size at most n_{i+j}. For convenience, we refer to $d(s_j)$ for $j \in \omega$ as the *special* entries of d, and call $d(x)$ for all other x as the *non-special* entries. Note that we do not truly need to differentiate between the two; this definition is made purely for ease of argument. Define the primitive recursive scheme computing q_e from d as follows. Given $j \in \omega$, compute $d(x)$ for each $x \geq s_i$ until we have computed $d(s_i + 2(j+1) + 1)$. Since each entry is either special or non-special, this process guarantees that we must either have computed d on at least $j + 2$ many special entries, or at least $j + 2$ many non-special entries.

We shall argue that by the sub-stage given by the maximum of the e-th bit of the various $d(x)$ computed by the procedure above, an appropriate q_e image for a_j must have been enumerated into B_e. Consider the cases as follows:

- Suppose that we have computed at least $j + 2$ many special entries of d. That is, we have computed d on the entries $s_i < s_{i+1} < \cdots < s_{i+j+1} \leq s_i + 2(j+1) + 1$. Since a_j has size at most n_{i+j}, and all such orbits must have been enumerated into A before stage s_{i+j+1}, then it follows that the e-th bit of $d(s_{i+j+1})$ encodes a sub-stage large enough such that $A[s_{i+j+1}]$ has become a substructure of B_e, providing an appropriate q_e image for a_j.
- Suppose that the previous case does not hold. That is, there exists $k \leq j$ such that $s_{i+k} \leq s_i + 2(j+1) + 1 < s_{i+k+1}$. From the assumption earlier that there is at least one orbit of size n_l for each $l \in \omega$, a_j must have size $< n_{i+k+1}$. If a_j has size $< n_{i+k}$, then using the same argument as before, the sub-stage at which a q_e image for a_j is enumerated into B_e happens before the sub-stage given by

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

the e-th bit of $d(s_{i+k})$. In addition, if a_j has size $> n_{i+k}$, then $t_{i+k+1}\downarrow$ and a_j must have been enumerated into A by stage t_{i+k+1}. Recall that if $t_{i+k+1}\downarrow$, then the e-th bit of $d(s_{i+k})$ is at least as big as the sub-stage at which $A[t_{i+k+1}]$ becomes a substructure of B_e. Therefore, we may find an appropriate q_e image for a_j by such a sub-stage.

We may thus assume that a_j has size exactly n_{i+k}. Applying the case assumption, we also obtain that we have computed d on at least $j+2$ many non-special entries, denoted by $x_0 < x_1 < \cdots < x_{j+1}$. Recall that on non-special entries, the e-th bit in $d(x)$ is a sub-stage at which B_e enumerates the $(x - s_l)$-th orbit of size n_l, where l is the largest such that $s_l < x$. Therefore, each of the e-th bits of $d(x_0), d(x_1), \ldots, d(x_{j+1})$, should encode the sub-stage at which some orbit of size n_l enters B_e for some $l \geq i$. Since a_j is the $(j+1)$-st orbit of size $\geq n_i$ to be enumerated into A, it follows that at least one of the e-th entries of $d(x_0), d(x_1), \ldots, d(x_{j+1})$, gives a sub-stage large enough at which an appropriate q_e image for a_j must have been enumerated into B_e.

Thus, provided that $B_e \cong A$, that is, the e-th bit of $d(x)$ is non-zero for all $x \geq e$, we may primitive recursively compute an isomorphism $q_e : A \to B_e$ and its inverse using d. Theorem 3.1 is proved. ⊣

§4. Degrees of punctual categoricity. In this section, we show that in every non-zero c.e. Turing degree, there exists a PR-degree that is low for punctual isomorphism and a PR-degree that is a degree of punctual categoricity. We split the proof into Theorems 4.1 and 4.2.

THEOREM 4.1. *Let* $\mathbf{d}$ *be a non-zero c.e. Turing degree, then there exists* $f \equiv_T \mathbf{d}$ *such that* f *is low for punctual isomorphism.*

PROOF. Let W be a non-recursive c.e. set and let $\{\Psi_i\}_{i\in\omega}$ and $\{A_n\}_{n\in\omega}$ be listings of all primitive recursive schemes and all punctual structures, respectively. We construct a total function $f : \omega \to \{0,1\}$ such that $f \equiv_T W$ and satisfying the following requirements:

$R_{i,j,m,n}$: If $\Psi_i^f : A_m \to A_n$ and $\Psi_j^f : A_n \to A_m$ are isomorphisms such that $\Psi_i^f \circ \Psi_j^f$ is the identity map, then A_m and A_n are punctually isomorphic.

An initial description: To construct $f \equiv_T W$, we use a standard permitting technique. Throughout the construction, there will be a set of *markers* $\{m_0 < m_1 < \ldots\} \subseteq \omega$ such that $f(m_i) = W(i)$, and a marker m_i may be moved during stages at which some $j < i$ is enumerated into W. This ensures that $W \leq_T f$ (to be proved later). To obtain $f \leq_T W$, we will only change $f(x)$ at stage s if some $y \leq x$ is enumerated into W at stage s.

We now turn our attention to the requirements $R_{i,j,m,n}$. Let $i, j, m, n \in \omega$ be given. The oracle f begins as the all 0 function. For a single requirement $R_{i,j,m,n}$, we maintain a *pointer* p which tracks the value at which $R_{i,j,m,n}$ is currently searching for an *error*; let $f^* := (f \upharpoonright p)^\frown 1^\omega$ and check if both of the following holds:

- $\Psi_i^{f^*} \circ \Psi_j^{f^*}$ is the identity.
- $\Psi_i^{f^*}$ and $\Psi_j^{f^*}$ are both injective and preserve all functions and relations in the signature of A_m and A_n.

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

If no error is ever found at this value of p, then $\Psi_i^{f^*}$ must be a punctual isomorphism from A_m to A_n, since f^* is clearly primitive recursive. On the other hand, if one of the conditions does not hold, it must be witnessed at some finite stage s with the oracle $f^* \upharpoonright s$. By changing $f \upharpoonright s$ to $f^* \upharpoonright s$, and provided that $f \upharpoonright s$ never again changes, this failure of one of the above conditions will be preserved, thus ensuring that $R_{i,j,m,n}$ is satisfied.

In order to stay consistent with the permitting strategy, *permission* must be provided by W for $f \upharpoonright s$ to change. That is, some $x \leq p$ has to enter W at some stage after the error for $R_{i,j,m,n}$ is found in order for $f(y)$, where $p \leq y < s$, to change. For a fixed p, permission might never be obtained. To circumvent this, $R_{i,j,m,n}$ should search for errors at multiple locations. More specifically, for the strategy of $R_{i,j,m,n}$, we also attempt to define a computable function g_e, where $e = \langle i, j, m, n \rangle$ as follows. At each stage s of the construction, p will 'point' to some marker m_x. Once an error for $R_{i,j,m,n}$ has been found at (the current value of) m_x, define $g_e(x)$ to also be the current value of $W(x)$. Then 'point' p to the marker m_{x+1} and repeat the process. The function g_e as defined is clearly computable, and provided that errors are always successfully found at each m_x, g_e will also be total. Since W is non-recursive, there must be some x at which x enters W at a stage after an error was found for $R_{i,j,m,n}$ at m_x, thus allowing us to *act* for $R_{i,j,m,n}$ by changing the oracle f.

As the strategy of each requirement $R_{i,j,m,n}$ only depends on whether an error can be found, we use the pairing function, letting $e = \langle i, j, m, n \rangle$, and simply refer to $R_{i,j,m,n}$ as R_e. Similarly, p_e and g_e will, respectively, be the pointer and computable function associated with the strategy for R_e.

The general strategy and injury: The priority of the requirements will be arranged as $R_0, R_1, \ldots$. Since the values of the markers are dynamic, when we say that p_e *points* to m_x, this means that the value of p_e copies m_x as it changes. At each stage of the construction, R_e will always be searching for errors using $f_e := (f \upharpoonright p_e)^\frown 1^\omega$. Once an error is found, define $g_e(x)$ to be the current value of $W(x)$ where x is such that p_e currently points to m_x. Also let p_e now point to m_{x+1}. As discussed in the initial description of the strategy, there is either some x for which no error is ever found for R_e at x or x eventually enters W, providing permission for R_e to act.

While the strategy for R_e is being pursued, one of the following could happen:

- The value of m_x at which R_e previously found an error changes.
- The value of m_x at which p_e is currently pointing changes.
- The oracle changes value at y for some $y < p_e$.

In the above scenarios, it is evident that any potential satisfaction found for R_e may now become undone, and we say that R_e is *injured*. However, we shall show that in each of the above scenarios, progress in the strategy of a higher priority $R_{e'}$ must have been made. To give a brief description of why this is so, recall that m_x changes only if some $x' < x$ enters W. In particular, W has just provided permission to change $f(y)$ for any $y \geq x'$. That is, any $R_{e'}$ which had earlier found an error at $m_{x'}$ now has permission to act. As long as we arrange the construction such that such an $R_{e'}$ must be of higher priority or $e' = e$, it is clear that the potential injury is finite. To this end, throughout the construction, $p_e < p_{e'}$ if $e < e'$. In addition, for each x such that $p_e < m_x < p_{e'}$, it must be that $R_{e'}$ has found an error at m_x.

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

Construction: During the construction, each requirement R_e will be declared either satisfied or unsatisfied. At the start of the construction, for each $e \in \omega$, let $m_e[0] = e$, point each p_e at m_e, and declare R_e to be unsatisfied. Also define $f(x) = 0$ for all $x \in \omega$. At each stage s of the construction, do the following.

Step 1: For each $e \leq s$ such that R_e is currently unsatisfied, search for an error for R_e at p_e with the oracles $(f \upharpoonright p_e)^\frown 1^{s-p_e}$, and run each computation till stage s.

Step 2: Let e be the least such that R_e has found an error at p_e and R_e is currently unsatisfied. If no such e exists, then proceed to Step 3. Otherwise, let x be such that p_e is currently pointing at m_x. Then define $g_e(x) = W(x)[s]$, and let p_e now point to m_{x+1}. For each $e' > e$, declare $R_{e'}$ to be unsatisfied, define $g_{e'}(y) \uparrow$ for each $y \in \omega$, and point $p_{e'}$ to $m_{x+1+e'-e}$. For convenience, we refer to this action as *initialising* $R_{e'}$.

Step 3: Let e be the least such that R_e has found an error at $m_x[s]$ for which x has just entered W. Then define $f \upharpoonright s$ to be $(f \upharpoonright m_x)^\frown 1^{s-m_x}$, declare R_e to be satisfied, and initialise $R_{e'}$ for each $e' > e$. Also for each $x' \geq x$, define $m_{x'}[s+1]$ to be the least number $\geq s$ and $\neq m_{x''}[s+1]$ for any $x'' < x'$. Notice that f on all such inputs is currently 0. For each $x' \geq x$ that is currently contained in W, define $f(m_{x'}[s+1]) = 1$.

Verification: To verify that the construction works, we prove the following statements:

(i) For each $x \in \omega$, the value of $m_x := \lim_s m_x[s]$ never changes after the stage s for which $W[s] \upharpoonright x = W \upharpoonright x$, and $f(m_x) = W(x)$.

(ii) For each $e \in \omega$, there is some $x_e \in \omega$ such that p_e points at m_{x_e} for cofinitely many stages.

Let $x \in \omega$ be given. Fix the least stage s such that $W[s] \upharpoonright x = W \upharpoonright x$. The only action in the construction which possibly changes the value of m_x is in Step 3. In particular, if $m_x[s+1] \neq m_x[s]$, then at stage s, some $x' < x$ must have been enumerated into W. Since s is a stage at which $W[s] \upharpoonright x = W \upharpoonright x$, this cannot happen after stage s, and thus $\lim_s m_x[s] = m_x[s+1]$. To see why $f(m_x) = W(x)$, consider the following cases:

- There is some stage t such that x enters W at stage t. If $t \geq s$, then at stage t, Step 3 of the construction ensures that we define $f(m_x[t+1]) = 1$. Since the value of m_x never changes after stage s, it follows that $f(m_x) = f(m_x[t+1]) = 1$. On the other hand, if $t < s$, then it must be that $s > 0$, and at stage s, some $x' < x$ enters W. Then the construction would have defined $f(m_x[s+1]) = 1$ since $x \in W[t] \subseteq W[s]$ and $x > x'$. Just as before, since the value of m_x never again changes after stage $s+1$, $f(m_x) = f(m_x[s+1]) = 1$.
- If $x \notin W$, then we claim that $f(m_x) = 0$. Observe that whenever some $x' < x$ enters $W[t]$, $m_x[t+1]$ would be defined to be a fresh value; thus $f(m_x[t+1])[t] = 0$. This implies that $f(m_x) = f(m_x[s+1])[s] = 0$ since no $x' < x$ ever enters W after stage s.

Using (i), we may easily show that $f \equiv_T W$. Given f and x, suppose that $m_{x'}$ has been computed for each $x' < x$, and search for the stage s such that $W[s] \upharpoonright x = \{x' \mid x' < x \text{ and } f(m_{x'}) = 1\}$. For such a stage s, it must be that $m_x[s+1] = m_x$,

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

and we may then retrieve $W(x)$ by computing $f(m_x)$. To compute $f(x)$, since $f(x)$ possibly changes only when some $y \leq x$ enters W, at the stage s such that $W[s] \upharpoonright (x+1) = W \upharpoonright (x+1)$, $f(x)[s+1] = \lim_s f(x)[s]$.

From Steps 2 and 3 of the construction, it is evident that the markers m_x at which each p_e points is increasing (in x). Thus, if p_e does not point at a single marker for cofinitely many stages, then p_e points at each marker for only finitely many (possibly 0) stages. Let $e \in \omega$ be the least such that this holds. Following the construction, a pointer p_e only changes the marker it points at due to one of the following reasons:

- An error for R_e was found at m_x at which p_e is currently pointing. Then p_e subsequently points at m_{x+1}. (See Step 2 of the construction.)
- R_e has acted (see Step 3).
- R_e was initialised.

R_e is only initialised if there is some $e' < e$ such that an error for $R_{e'}$ was found (see Step 2), or $R_{e'}$ has acted (see Step 3). In either of these situations, the marker $p_{e'}$ is pointing at changes. By the assumption that e is the least for which the marker p_e points at changes infinitely often, we may assume that past some finite stage s^*, p_e cannot change due to R_e being initialised. After such a stage, p_e changes only if it has acted or if an error is found at the current marker p_e is pointing at.

If R_e has acted, then it must have been declared to be satisfied. Furthermore, as no other higher priority requirement acts after s^*, Step 2 of the construction never again considers R_e. In other words, once R_e acts after stage s^*, p_e never again changes the marker it points at.

Finally, we consider the possibility that R_e never acts after stage s^*. Then the only way that p_e changes the marker it points at infinitely often is if errors are found at each marker that p_e points at. In addition, R_e never receives permission to act upon these errors; there is some x such that for all $x' \geq x$, $g_e(x') \downarrow = W(x')$, contradicting the assumption that W is non-recursive.

A quick analysis of the construction allows one to conclude that if p_e stabilises, then R_e must remain in the satisfied state for cofinitely many stages. In addition, this means that either some error for R_e was found and acted upon, or there is some x such that no error for R_e is ever found at m_x. Thus, each R_e is met. Theorem 4.1 is proved. ⊣

THEOREM 4.2. *Let* $\mathbf{d}$ *be a non-zero c.e. Turing degree, then there exists* $g \equiv_T \mathbf{d}$ *such that* g *is a degree of punctual categoricity.*

PROOF. Let W be a non-recursive c.e. set, $\{p_e\}_{e\in\omega}$ be a listing of primitive recursive functions, and let $\{B_e\}_{e\in\omega}$ be a listing of the punctual structures in the language of our structure B. We construct $g \equiv_T W$ satisfying the following requirements:

G : There exists punctual $B' \cong B$ such that if $f : B \to B'$ is an isomorphism, then $f \oplus f^{-1} \geq_{pr} g$.

$Q_{m,n}$: If $B_m \cong B_n \cong B$, then there exists $h : B_m \to B_n$, an isomorphism, such that $g \geq_{pr} h \oplus h^{-1}$.

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

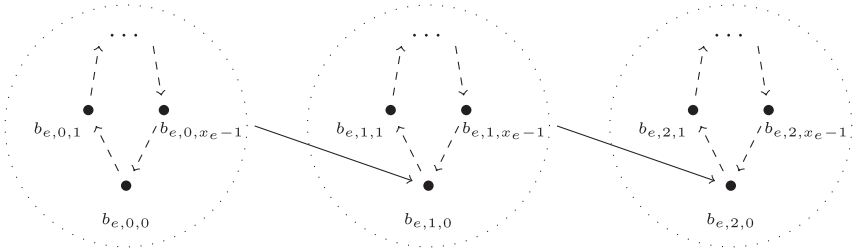


FIGURE 1. The e-th component of B. The function arrows for R and P are not shown here. The solid arrows represent the function S, and the dashed arrows represent C.

For each $x \in \omega$, define $g(\langle 0, x\rangle)$ to be the stage at which x enters W, and -1 if no such stage exists. For each $e > 0$ and each $x \in \omega$, also define $g(\langle e, x\rangle)$ to be the stage at which $p_{e-1}(x)\downarrow$. It is evident that $g \equiv_T W$.

An initial description of B: We adopt the pressing strategy from [26]; we construct B so that each potential punctual structure in the same language is either g-punctually isomorphic to B or not isomorphic at all. The language of our structure will consist of four unary function symbols S, P, R, and C defined below. In the e-th *component* of our structure, we will have elements which form a *spine*, given by $b_{e,i,0}$ for the various $j \in \omega$, and a distinguished element, $b_{e,0,0}$ which is the *root* (illustrated in Figure 1).

- For each element along the spine, $b_{e,i,0}$, the function C generates x_e*-cycles*, consisting of finitely many elements $b_{e,i,j}$, for each $j < x_e$ with the property that $C(b_{e,i,x_e-1}) = b_{e,i,0}$ and $C(b_{e,i,j}) = b_{e,i,j+1}$ for each $j < x_e - 1$.
- The range of the map S consists only of elements along the spine. More specifically, for each i and $j < x_e$, $S(b_{e,i,j}) = b_{e,i+1,0}$.
- P is the identity map on all elements of the form $b_{e,i,j}$.
- Finally, R maps all elements within the e-th component to $b_{e,0,0}$, the root. This ensures that in any potential copy of B, the roots of each component must be enumerated quickly after any element within that same component is enumerated.

To aid in the strategy for G and the various $Q_{m,n}$, we shall keep the components 'sufficiently' different by keeping the cycle sizes of each component different.

The pressing strategy: Except for stages where we 'switch' (see strategy for R), the structures B and B' will be enumerated in exactly the same way. In much the same spirit as the original pressing argument, we want to 'force' the structures B_e to remain as substructures of B and B' at each stage. In general, the $2e$-th and $(2e+1)$-st components of B will depend on the structures B_n for each $n \leq e$. At each stage of the construction, only one component is not *closed*: there is some element in the component such that S is yet to be defined on that element.

For simplicity, first consider the 0-th component. Set the cycle size for the 0-th component as 1. While waiting for B_0 to enumerate elements, continue enumerating the elements $b_{0,i,0}$ of the spine of the 0-th component and defining $C(b_{0,i,0}) = b_{0,i,0}$.

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

When some element is enumerated into B_0, compute $C^m R(b)$ for each $m \leq M_1$ (where $M_1 \geq 1$ to be defined later).

- If $CR(b) \downarrow = R(b)$, then it means that the structure B_0 has enumerated the root of the 0-th component.
- If the least $m > 0$ for which $C^m R(b) \downarrow = R(b)$ is not 1, then B_0 has revealed some cycle size not yet enumerated into B. By never enumerating such a cycle size into B, we may then guarantee that $B_0 \ncong B$.
- If such an $m > 0$ does not exist, that is, $C^m R(b) \downarrow \neq R(b)$ for each $0 < m \leq M_1$, then we may intuitively think of this as the situation in which B_0 has revealed some 'large' cycle size. While we may not directly obtain that $B_0 \ncong B$ in this case, by some careful arrangement of the cycle sizes of each subsequent component, we may avoid the eventual size of the cycle of $R(b)$.

Once one of the above is witnessed, we may then *close* the 0-th component by defining $S(b_{0,i,0}) = b_{0,i,0}$, where i is the largest index such that $b_{0,i,0}$ currently exists in B. The idea here is that in the first case, we have successfully made B_0 'copy' our structure B, and hence may proceed to enumerating other components. In the second and third cases, we continue enumerating the 0-th component until some guarantee that $B_0 \ncong B$ is obtained before closing the 0-th component and starting new components. In particular, there is either some element $b \in B_0$ such that $R(b)$ has a different cycle size than anything enumerated into B thus far, or has cycle size at least M_1.

Let $e > 0$ be given and suppose that M_e has been defined with the property there is at least one (non-zero) size $x_e < M_e$ that is available. In addition, for each $n \leq \lfloor e/2 \rfloor$, one of the following holds:

- For each $e' < e$, the root of the e'-th component has been enumerated into B_n; there is some $b \in B_n$ such that the least $m > 0$ for which $C^m R(b) \downarrow = R(b)$ is such that $m = x_{e'}$.
- There is some $b \in B_n$ such that the least $m > 0$ for which $C^m R(b) \downarrow = R(b)$ is such that $m \neq x_{e'}$ for any $e' < e$. Then such a size m must have been declared unavailable, and we shall refer to such B_n as *inactive*.
- If B_n is not inactive, but there is some $b \in B_n$ such that for each $0 < m \leq M_e$, $C^m R(b) \downarrow \neq R(b)$, then we say that B_n is *pending* and call b the $e-1$-*witness*.

The intuition is that for B_n which are inactive, we already have that $B_n \ncong B$, while the B_n which are active are those that we still have to monitor during the construction. For those B_n which are pending, there is still some 'mystery' element $b \in B_n$ not revealed to be part of any component enumerated into B thus far. The idea is that we do not want such an element to be able to mimic some component enumerated into B at a later stage. In building the e-th component, we need to take into account all such structures and ensure that we only enumerate cycles with sizes smaller than $R(b)$ for the various witnesses b.

Begin the e-th component by enumerating the elements $b_{e,i,0}$ and letting the cycle sizes be x_e: for each $j < x_e - 1$, let $C(b_{e,i,j}) = b_{e,i,j+1}$ and $C(b_{e,i,x_e-1}) = b_{e,i,0}$. We shall only close the e-th component when, for each $n \leq \lfloor e/2 \rfloor$, where B_n is not inactive, either B_n has enumerated the root of the e-th component, or if B_n has a e-witness b. Recall that this means that for each $0 < m \leq M_{e+1}$ (sufficiently large, to

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

be defined shortly), $C^m R(b) \downarrow \neq R(b)$. As before, while waiting for these conditions to be met, we continue extending the spine of the e-th component to keep B (and B') punctual.

- If B_n was previously declared pending, then it has a $(e-1)$-witness b. That is to say, $R(b)$ cannot possibly have cycle size $x_e < M_e$, since B_n has already shown that $C^{x_e} R(b) \downarrow \neq R(b)$. Continue computing $C^m R(b)$ for the $(e-1)$-witness b. This process either produces some $M_e < m \leq M_{e+1}$ such that $C^m R(b) \downarrow = R(b)$, or we find that for each $0 < m \leq M_{e+1}$, $C^m R(b) \downarrow \neq R(b)$. If the former holds, then declare B_n to be inactive and declare the size m to be unavailable (recall that this means that such a size is never enumerated after this stage). Furthermore, note that $m \neq x_{e'}$ for any $e' \leq e$; there are no cycles of size m in B currently. If the latter holds, then observe that b is now a e-witness for B_n (and trivially still a $(e-1)$-witness for B_n). In summary, for each pending B_n, it either remains pending, or becomes inactive.
- If B_n is neither inactive nor currently pending, then for each element b enumerated into B_n, compute for each $0 < m \leq M_{e+1}$, $C^m R(b)$. If for each $m \leq M_{e+1}$, $C^m R(b) \downarrow \neq R(b)$, then declare B_n to be pending with the e-witness b. We may thus assume that there is some $0 < m \leq M_{e+1}$ for which $C^m R(b) \downarrow = R(b)$. Let m be the least where this holds. If $m \neq x_{e'}$ (or $y_{e'}$, to be defined later) for any $e' \leq e$, then declare m to be unavailable, and B_n to be inactive. For the same reasons as before, B_n cannot possibly be isomorphic to B. Recursively suppose that the e'-th component for each $e' < e$ is finite. That is, there are only finitely many elements $b' \in B$ such that $R(b')$ has cycle size $x_{e'}$ (or $y_{e'}$) for some $e' < e$. In particular, B_n must eventually enumerate some element b for which $R(b)$ does not have cycle size $x_{e'}$ for any $e' < e$, otherwise, we will again discover that $B_n \not\cong B$ (and can declare B_n to be inactive). That is, if B_n is not declared to be pending or inactive, then it must, in finite time, enumerate an element b such that the least $m > 0$ for which $C^m R(b) \downarrow = R(b)$ is exactly x_e.

Evidently, after starting the e-th component in B, for each $n \leq \lfloor e/2 \rfloor$, within finite time, each B_n must either reveal a e-witness, become inactive, or enumerate the root of the e-th component. Once such a stage is reached, then we close the e-th component by defining $S(b_{e,i,j}) = b_{e,i,0}$ for each $j < x_e$ and where i is the unique index for which $S(b_{e,i,j})$ has yet to be defined. Roughly speaking, since each structure potentially only 'blocks' one size, and each component uses only one size, it is not hard to see that as long as $M_{e+1} > (e+1) + (\lfloor e/2 \rfloor + 1) + 1$, we may always successfully find $x_{e+1} < M_e$ that is available to be the cycle size of the $(e+1)$-st component. As shall be seen shortly there are some additional factors that contribute to the definition of M_e, thus we momentarily delay giving a definition.

The strategy for G: The idea here is to use the image of the root of each component under a given isomorphism from B to B' to encode the values of g. However, since each component depends on a growing number of punctual structures, there cannot be a primitive recursive way to list the indices of the roots of each component. To overcome this, we shall dedicate the components with index $2e$ to encode a stage large enough such that B has closed the $(2e+2)$-nd component, and necessarily also closed the $(2e+1)$-st component. Roughly speaking, to obtain the indices of

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

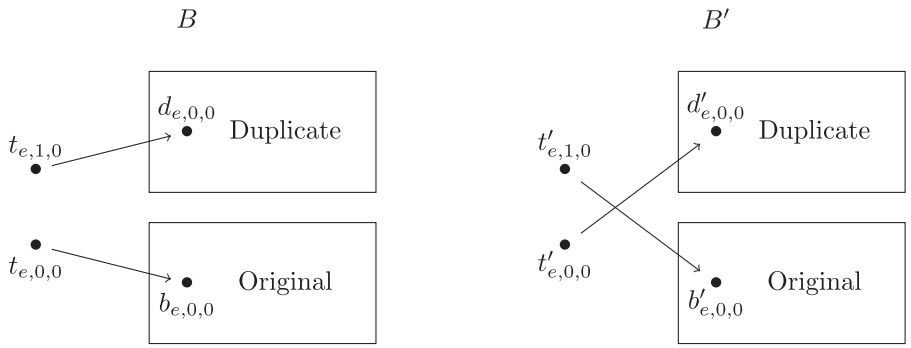


FIGURE 2. Switching in the e-th component. The arrows represent the function P.

$b_{e',0,0}$ for each $e' \leq 2e$, compute the given isomorphism on $b_{0,0,0}$, obtaining the index of $b_{2,0,0}$, and repeating till we obtain the index of $b_{2e,0,0}$. Observe that this process is primitive recursive in the given isomorphism. The elements $b_{2e+1,0,0}$ will be used to encode the value of g.

We say that a component is *ready* if the following holds:

- If the component is the $2e$-th component and the $(2e+2)$-nd component has closed in the structure B (and thus also B').
- If the component is the $(2\langle 0, x\rangle + 1)$-st component and x has entered W.
- If the component is the $(2\langle e+1, x\rangle + 1)$-st component and $p_e(x)$ has converged.

Suppose that all components in B and B' are currently closed. For the component with the lowest index, say e, that is ready, we perform the *switch* as follows. Enumerate an exact duplicate of the e-th component into B, containing the elements $d_{e,i,j}$ for each i, j where $b_{e,i,j}$ is currently an element in the e-th component. In particular, the collection of all the $b_{e,i,j}$ are indistinguishable from the collection of $d_{e,i,j}$ up to isomorphism. Do the same in B'. Note that by convention, each of these $d_{e,i,j}$, and also $d'_{e,i,j} \in B'$ has indices larger than the current stage number. Thus, we would like to 'force' any isomorphism from $B \to B'$ to map the elements from the original component in B to the duplicated component in B'. To achieve this, we enumerate a *tail* to join the original e-th component with the duplicate e-th component (see Figure 2).

Since the duplicate and original e-th component are identical up to isomorphism, we have the freedom to attach the tail differently in B and B' as shown in the figure while maintaining $B \cong B'$. Evidently, $t_{e,0,0}$ and $t_{e,1,0}$ should be distinguishable up to isomorphism in order to ensure that isomorphisms from B to B' has to map $b_{e,0,0}$ to $d'_{e,0,0}$ (with large index) in B'. The tail will consist of elements labelled $t_{e,i,j}$ (in B) and the maps S, P, R, C are defined on them as follows:

- Let e' be the largest index for which the e'-th component currently exists in B. Pick a size $0 < y_e < M_{e'+1}$ different from any cycle size in B thus far which is still available (we shall argue later that such sizes must exist by choosing a sufficiently large $M_{e'+1}$). All cycles of elements in the tail shall have size y_e.

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

- As shown in the figure define in B, $P(t_{e,0,0}) = b_{e,0,0}$, $P(t_{e,1,0}) = d_{e,0,0}$, and define in B', $P(t'_{e,0,0}) = d'_{e,0,0}$, and $P(t'_{e,1,0}) = b'_{e,0,0}$. For the rest of the elements in the tail, P is again the identity map. This will be the only place in which B and B' 'locally' differ.
- R of any element of the tail maps to $t_{e,0,0}$ in B and $t'_{e,0,0}$ in B'.
- For any $j < y_e$, and any i, define $S(t_{e,i,j}) = t_{e,i+1,0}$, while waiting for the tail to close.

Observe that whenever any element t of the tail is enumerated into B_n, the elements $R(t)$ and $SR(t)$ should also be enumerated 'quickly' into B_n. In addition, the tail also 'joins' up in primitive recursive (in B_n) time to the original and duplicated e-th component. In much the same way as before, we apply the pressing strategy to ensure that either $B_n \not\cong B$ (declared inactive or pending), or B_n enumerates the tail. In addition, by choice of y_e, y_e is strictly smaller than the eventual cycle sizes for $R(b)$ of any e'-witness b of B_n where $n \leq \lfloor e'/2 \rfloor$, and different from any size declared unavailable thus far. Once the copies of $t_{e,0,0}$ have been enumerated in each B_n for each $n \leq \lfloor e/2 \rfloor$, we close the tail in the same way as before: for the unique i on which $S(t_{e,i,0})$ is yet to be defined, let $S(t_{e,i,j}) = t_{e,i,0}$ for each $j < y_e$.

We now explain how the various M_e may be chosen. Recall that each structure potentially only declares one size to be unavailable, and the original components use only exactly one size each. Similarly, each tail uses also only exactly one size. Thus, by picking M_{e+1} to be larger than the sum of the total number of potential tails, $e+1$, the number of components, $e+1$, and the number of structures, $\lfloor e/2 \rfloor + 1$, it is guaranteed that there must always be available sizes as required. More specifically, pick $M_{e+1} = 4(e+1) > 2(e+1) + (\lfloor e/2 \rfloor + 1) + 1$.

The strategy for $Q_{m,n}$: If $m = n$, then the requirement is trivially satisfied by the identity map. We consider, without loss of generality, the case where $m < n$. The main challenge is in recovering the stage at which both the original and duplicate components are enumerated into both B_m and B_n. Evidently, if we do not wait till such a stage, any attempt at defining an isomorphism may fail; there is no way of knowing if the original component in B_m maps to the original or duplicate component in B_n.

Recall that the amount of time B spends enumerating the e-th component depends on the structures B_k for each $k \leq \lfloor e/2 \rfloor$. Since each of these structures are punctual, and g primitive recursively computes every primitive recursive function ($g(\langle e+1, x \rangle)$ is the stage at which $p_e(x)$ converges), then we should be able to recover the stages at which each component in B closes. In much the same vein, we may also recover the stage at which the tail is fully enumerated into B and similarly, each B_k. (This will be verified formally later.) Thus, we may obtain stages large enough at which B_m and B_n have completed enumerating the full 'local' structure, allowing us to define the isomorphism in the obvious way.

Construction: At each stage of the construction, there will be exactly one component, or tail of a component which is not closed in B and B'; there is exactly one pair e, i, such that $S(b_{e,i,0})$ and $S(b'_{e,i,0})$ have yet to be defined. Recall that M_e is defined as $4e$ for each e and that an element b of B_n is an e-witness if for each $0 < m \leq M_{e+1}$, $C^m R(b) \downarrow \neq R(b)$. We also say that B_n is pending if there is some e-witness for B_n.

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

During stage s, let e be the least such that one of the following holds:

(i) All components and their tails currently in B are closed, and $b_{e,0,0}$ has yet to be enumerated into B.
(ii) The e-th component is not closed in B.
(iii) All components and their tails currently in B are closed, and the e-th component is ready.
(iv) The tail of the e-th component is not closed in B.

Case (i): If (i) holds, then recursively assume that for each $n \leq \lfloor (e-1)/2 \rfloor$, if B_n has been declared pending, then it has a $(e-1)$-witness. Let $0 < x_e < M_{e+1}$ be the least size which is currently available. Use x_e as the cycle size for the e-th component and declare it unavailable (it will not be used for any other component or tail). Begin the e-th component in both B and B', enumerating $b_{e,0,0}$ and $b'_{e,0,0}$ with cycle sizes x_e, respectively. Leave $S(b_{e,0,j})$ and $S(b'_{e,0,j})$ undefined for each $j < x_e$.

Case (ii): If (ii) holds, then let i be the largest index such that $S(b_{e,i,0})$ has yet to be defined. For each $n \leq \lfloor e/2 \rfloor$, where B_n is not inactive, check if the following holds:

- If B_n is pending, then b has a e-witness.
- If B_n is not pending, then it has enumerated some element b such that the least $m > 0$ for which $C^m R(b) \downarrow = R(b)$ is exactly x_e.

If the above is true for all B_n which are not inactive, then close the e-th component by defining $S(b_{e,i,j}) = b_{e,i,0}$ in B and $S(b'_{e,i,j}) = b'_{e,i,0}$ in B' for each $j < x_e$. If the above fails for some $n \leq \lfloor e/2 \rfloor$, then continue extending the e-th component by enumerating $b_{e,i+1,0}$ with cycle size x_e and defining $S(b_{e,i,j}) = b_{e,i+1,0}$. Also continue computing $C^m R(b)$ for each $b \in B_n$ and each $m \leq M_{e+1}$. As discussed previously one of the following must happen:

- B_n enumerates the root of the e-th component: B_n enumerates an element $R(b)$ and the least $m > 0$ for which $C^m R(b) \downarrow = R(b)$ is exactly x_e.
- Some e-witness for B_n is found, then we may declare B_n pending (if it was not previously pending) and proceed to the next required action.
- $C^m R(b) \downarrow = R(b)$ for some $0 < m \leq M_{e+1}$ but $m \neq x_e$. There can only possibly be finitely many such b found in B_n such that $m = x_{e'}$ or $y_{e'}$ for some $e' < e$, after which B_n will be declared inactive. The only other possibility is that B_n enumerates some b for which the least $0 < m \leq M_{e+1}$ where $C^m R(b) \downarrow = R(b)$ is different from all $x_{e'}$ and $y_{e'}$ for $e' < e$. If such a b is found, then declare m to be unavailable, and B_n to be inactive.

That is, we eventually close the e-th component, and when we do, all currently pending B_n have a e-witness.

Case (iii): If (iii) holds, then let e' be the largest such that $b_{e',0,0}$ has been enumerated into B. Pick $y_e < M_{e'+1}$ to be the least size that is available and subsequently declare it unavailable (this ensures that y_e is different

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

from all earlier cycles and all future cycles in different components or tails). Perform the switch for the e-th component by enumerating $t_{e,0,0}$, $t_{e,1,0}$, the duplicate e-th component, and their counterparts into B and B', respectively. Let the cycle size of $t_{e,0,0}$ and $t_{e,1,0}$ be y_e. For each $j < y_e$, define $S(t_{e,0,j}) = t_{e,1,0}$ and leave $S(t_{e,1,j})$. As discussed in the strategy for G, define for each $i < 2$, $P(t_{e,i,0}) = d_{e,i,0}$ in B, and $P(t'_{e,i,0}) = d'_{e,1-i,0}$ in B'.

Case (iv): If (iv) holds, then let i be the largest such that $S(t_{e,i,0})$ is currently undefined. Just as in Case (ii), we only close the tail when for each $n \leq \lfloor e/2 \rfloor$, where B_n is not inactive, the following holds:

- If B_n is pending, then b has a e'-witness, where e' is currently the largest for which $b_{e',0,0}$ exists in B.
- B_n has enumerated some element b such that the least $m > 0$ for which $C^m R(b) \downarrow = R(b)$ is exactly y_e.

For the same reasons as before, we eventually close the e-th tail, and when we do, all currently pending B_n have a e'-witness.

Verification: We first show that B and B' are punctual structures. The functions P, R, and C are all defined immediately whenever some element is enumerated into B or B'. The only function which is delayed is S when we do not want to close the e-th component or tail yet. However, the delay lasts only for a single stage as seen in Cases (ii) and (iv) of the construction: for the elements $b_{e,i,j}$ or $t_{e,i,j}$ for which S is undefined at the end of stage s, the e-th component or tail either closes at stage $s+1$, or a new element $b_{e,i+1,0}$ or $t_{e,i+1,0}$ is enumerated to serve as the S-image. Thus, B and B' are punctual.

Now we show that if B_n is ever declared inactive or pending, then $B_n \not\cong B$. If B_n is declared inactive at some stage, then one of the following must hold:

- There are distinct $b_0, b_1, \ldots, b_N$ and some x_e such that for each $i \leq N$, the least $m > 0$ for which $C^m R(b_i) = R(b_i)$ is exactly x_e and N is strictly larger than the total number of elements in the closed e-th component in B. Since each component has a distinct cycle size, and also different from all potential and existing tails, B_n cannot possibly be isomorphic to B.
- There is some element $b \in B_n$ such that the least $m > 0$ for which $C^m R(b) \downarrow = R(b)$ is different from the size of any cycle currently in B. From Case (ii) of the construction, if such a b is found, we must have defined m to be unavailable, and thus never enumerate any cycle of the same size into B.

In other words, if B_n is ever declared inactive, then it must be that $B_n \not\cong B$. We may thus suppose that B_n is never declared inactive and hence stays pending forever. From the construction, a structure B_n is first declared pending if for some e, an e-witness for B_n is found. The claim now is that this e-witness b for B_n eventually becomes an e'-witness for all e'.

Inductively assume that b is an e'-witness. That is, for each $0 < m \leq M_{e'+1}$, $C^m R(b) \downarrow \neq R(b)$. From Case (ii) of the construction, the $(e'+1)$-st component is only closed after any pending structure has obtained an $(e'+1)$-witness. It remains to show that b must become an $(e'+1)$-witness. Since $x_{e'+1} < M_{e'+1}$ and b is an e'-witness, then the cycle size of $R(b)$ (in B_n) cannot possibly mimic that of any cycle in the $(e'+1)$-st component of B. If there is some $0 < m \leq M_{e'+2}$ for which

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

$C^m R(b) \downarrow = R(b)$, then the least $m > 0$ where this holds is necessarily larger than $M_{e'+1}$ and thus bigger than any cycle size shown in B thus far. In particular, m would then be declared as unavailable and B_n would become inactive. That is, it must be that $C^m R(b) \downarrow \neq R(b)$ for each $0 < m \leq M_{e'+2}$, and thus, b must also be an $(e'+1)$-witness. Therefore, B_n cannot possibly be isomorphic to B as B_n contains an infinite component.

$Q_{m,n}$ is satisfied: If $m = n$, then the identity map is obviously a punctual isomorphism. Without loss of generality, we may assume that $m < n$. By the premise of the requirement, $B_m \cong B_n \cong B$, and thus, it suffices to produce an isomorphism on each component. In particular, we need only compute a stage at which B_m and B_n has fully enumerated both the original and duplicate components together with the tail.

We claim that given g and e, it is primitive recursive to find the stage at which the e-th component closes in B and B', and thus the total number of elements within the e-th component. Recall that the e-th component in B and B' closes exactly when there is some $b \in B_k$ for each $k \leq \lfloor e/2 \rfloor$ such that one of the following holds (see Case (ii) of the construction):

(i) The least $0 < l \leq M_{e+1}$ for which $C^l R(b) \downarrow = R(b)$ is exactly x_e (the cycle size of the e-th component).
(ii) The least $0 < l \leq M_{e+1}$ for which $C^l R(b) \downarrow \neq R(b)$ is different from $x_{e'}$ and $y_{e'}$ if they exist, for each $e' \leq e$. (B_n will then be declared inactive.)
(iii) For all $0 < l \leq M_{e+1}$, $C^l R(b) \downarrow \neq R(b)$. ($b$ is a e-witness for B_n.)

Since g primitive recursively computes all primitive recursive functions, and B_k is punctual, we may assume that we have access to the versions of S, P, R, and C in each B_k. In particular, using g, we apply the following procedure:

- First compute the stage at which B and B' closes the 0-th component, or equivalently, the stage at which some $b \in B_0$ is found satisfying one of (i), (ii), or (iii) for $e = 0$. This is clearly primitive recursive in g, as we may use g to compute the actions of the versions of the maps R, C of B_0, and we need only check if $CR(b) \downarrow = R(b)$ for the first b enumerated into B_0. In fact, g provides the stage at which $CR(b) \downarrow$, and at such a stage, we must have closed the 0-th component. In particular, the number of elements in the 0-th component is primitive recursive in g and 0.
- Suppose recursively that the number of elements in the original e'-th components of B and B' for all $e' < e$ may be computed primitive recursively in g and e'. Since each B_k is punctual, we may compute using g, the settling time of each of the functions of B_k on the first N many elements of B_k, where N is the total number of elements in the first e many original components of B and B'. After the delay given by this settling time, it is evident that B_k must have enumerated some element b that is not part of any of the e'-th component for any $e' < e$. By further computing the settling time (using g) for $C^{m+1} R(b)$ on such a b, we may then obtain the stage at which some $b \in B_k$ is discovered to satisfy one of (i), (ii), or (iii). Repeating this for each $k \leq \lfloor e/2 \rfloor$ allows us to then obtain the stage at which the e-th component is closed in B and B'. Thus, such a stage can be computed primitive recursively in g and e.

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

By applying a similar argument, we may further obtain primitive recursively, using g and e, a stage at which the copy of the e-th component in B_k is closed or that B_k has been declared inactive or pending for each $k \leq \lfloor e/2 \rfloor$. Similarly, it is primitive recursive in g and the stage, say s_e, at which $t_{e,0,0}$ is enumerated into B to compute the stage at which the e-th tail is closed in B.

We now show that s_e may be obtained primitive recursively using g and e. If e is even, then recall that the e-th component becomes ready at the stage at which the $(e+2)$-nd component closes. Applying the fact above allows us to conclude that this can be computed primitive recursively using g and e. If e is odd, then the e-th component becomes ready at the stage given by $g((e-1)/2)$. Once the stage at which a component becomes ready is obtained, we need only further compute the stages at which any component of smaller index completes switching, or equivalently, the stages at which the tails of smaller index close. This is clearly primitive recursive in g and the stage at which the e-th component becomes ready. In conclusion, given g and e, we may primitive recursively compute a stage large enough such that B_k for any $k \leq \lfloor e/2 \rfloor$ has enumerated every element of the original and duplicate e-th components, and also the e-th tail. Applying this for every $e \geq 2n+2$ allows us to conclude that g may compute an isomorphism between B_m and B_n for each component with index $e \geq 2n+2$. Since each component and tail is finite, we may non-uniformly fix the isomorphism between B_m and B_n for the components with index $< 2n+2$. Thus, $g \geq_{pr} h \oplus h^{-1}$ for some $h : B_m \to B_n$ an isomorphism.

Computing $g(x)$: Let $f : B \to B'$ be an isomorphism. To show that G is satisfied, we compute $g(x)$ for a given x as follows. For each $e \leq x$, compute $f(b_{2e,0,0})$. Since the tail is rigid, f must map $t_{2e,0,0}$ in B to $t'_{2e,0,0}$ in B'. In particular, $f(b_{2e,0,0}) = fP(t_{2e,0,0}) = Pf(t_{2e,0,0}) = P(t'_{2e,0,0}) = d'_{2e,0,0}$. From Case (iii) of the construction, $d'_{2e,0,0}$ is enumerated into B' only at a stage after the $2e$-th component is ready; this is a stage at least as large as the stage where the $(2e+2)$-nd component is closed in B. Thus, $f(b_{2e,0,0})$ provides a stage larger than the index of $b_{2e+2,0,0}$ and necessarily also $b_{2e+1,0,0}$. Taking $e = x$, we may thus obtain the index of $b_{2x+1,0,0}$ primitive recursively in f. For similar reasons as before, $f(b_{2x+1,0,0}) = fP(t_{2x+1,0,0}) = Pf(t_{2x+1,0,0}) = P(t'_{2x+1,0,0}) = d'_{2x+1,0,0}$ which must have index larger than the value of $g(x)$. By enumerating W or computing the appropriate primitive recursive function up to the index of $d'_{2x+1,0,0}$ allows us to recover the exact value of $g(x)$. Theorem 4.2 is proved. ⊣

Acknowledgments. We also thank David Belanger for many helpful discussions.

Funding. K.M.N. was supported by the Ministry of Education, Singapore, under its Academic Research Fund Tier 2 (MOE-T2EP20222-0018) and Academic Research Fund Tier 1 (RG104/24).

The work of N. Bazhenov was supported by the Mathematical Center in Akademgorodok under the agreement No. 075-15-2025-349 with the Ministry of Science and Higher Education of the Russian Federation and also supported by the Science Committee of the Ministry of Science and Higher Education of the Republic of Kazakhstan (Grant No. AP26198617).

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

REFERENCES

[1] N. Bazhenov, *Effective categoricity for distributive lattices and Heyting algebras*. ***Lobachevskii Journal of Mathematics***, vol. 38 (2017), no. 4, pp. 600–614.

[2] N. Bazhenov, R. Downey, I. Kalimullin, A. Melnikov, and K. M. Ng, *Foundations of online structure theory*. ***Bulletin of Symbolic Logic***, vol. 25 (2019), no. 2, pp. 141–181.

[3] N. Bazhenov and I. Kalimullin, *Punctual categoricity spectra of computably categorical structures*. ***Algebra and Logic***, vol. 60 (2021), no. 3, pp. 223–228.

[4] N. Bazhenov, H. T. Koh, and K. M. Ng, *Primitive recursive categoricity spectra*, preprint. http://arxiv.org/abs/2603.08006

[5] N. A. Bazhenov, *Categoricity spectra of computable structures*. ***Journal of Mathematical Sciences***, vol. 256 (2021), no. 1, pp. 34–50.

[6] N. A. Bazhenov, I. S. Kalimullin, and M. M. Yamaleev, *Degrees of categoricity and spectral dimension*. ***Journal of Symbolic Logic***, vol. 83 (2018), no. 1, pp. 103–116.

[7] W. Calvert, D. Cenzer, V. Harizanov, and A. Morozov, *Effective categoricity of equivalence structures*. ***Annals of Pure and Applied Logic***, vol. 141 (2006), nos. 1–2, pp. 61–78.

[8] W. Calvert, D. Cenzer, V. S. Harizanov, and A. Morozov, *Effective categoricity of abelian p-groups*. ***Annals of Pure and Applied Logic***, vol. 159 (2009), nos. 1–2, pp. 187–197.

[9] D. Cenzer, R. Downey, J. B. Remmel, and Z. Uddin, *Space complexity of abelian groups*. ***Archive for Mathematical Logic***, vol. 48 (2009), no. 1, pp. 115–140.

[10] D. Cenzer, V. Harizanov, and J. B. Remmel, *Computability-theoretic properties of injection structures*. ***Algebra and Logic***, vol. 53 (2014), no. 1, pp. 39–69.

[11] D. Cenzer and J. B. Remmel, *Polynomial-time versus recursive models*. ***Annals of Pure and Applied Logic***, vol. 54 (1991), no. 1, pp. 17–58.

[12] ———, *Polynomial-time abelian groups*. ***Annals of Pure and Applied Logic***, vol. 56 (1992), nos. 1–3, pp. 313–363.

[13] B. F. Csima, M. Deveau, M. Harrison-Trainor, and M. A. Mahmoud, *Degrees of categoricity above limit ordinals*. ***Computability***, vol. 9 (2020), no. 2, pp. 127–137.

[14] B. F. Csima, J. N. Y. Franklin, and R. A. Shore, *Degrees of categoricity and the hyperarithmetic hierarchy*. ***Notre Dame Journal of Formal Logic***, vol. 54 (2013), no. 2, pp. 215–231.

[15] B. F. Csima and K. M. Ng, *Every Δ^0_2 degree is a strong degree of categoricity*. ***Journal of Mathematical Logic***, vol. 22 (2022), no. 3, p. 2250022.

[16] B. F. Csima and D. Rossegger, *Degrees of categoricity and treeable degrees*. ***Journal of Mathematical Logic***, vol. 24 (2024), no. 3, p. 2450002.

[17] B. F. Csima and J. Stephenson, *Finite computable dimension and degrees of categoricity*. ***Annals of Pure and Applied Logic***, vol. 170 (2019), no. 1, pp. 58–94.

[18] R. Downey, A. Melnikov, and K. M. Ng, *Foundations of online structure theory II: The operator approach*. ***Logical Methods in Computer Science***, vol. 17 (2021), no. 3, pp. 6:1–6:35.

[19] E. Fokina, I. Kalimullin, and R. Miller, *Degrees of categoricity of computable structures*. ***Archive for Mathematical Logic***, vol. 49 (2010), no. 1, pp. 51–67.

[20] J. Franklin and R. Solomon, *Degrees that are low for isomorphism*. ***Computability***, vol. 3 (2011), no. 2, pp. 73–89.

[21] A. N. Frolov, *Effective categoricity of computable linear orderings*. ***Algebra and Logic***, vol. 54 (2015), no. 5, pp. 415–417.

[22] S. S. Goncharov, *Degrees of autostability relative to strong constructivizations*. ***Proceedings of the Steklov Institute of Mathematics***, vol. 274 (2011), no. 1, pp. 105–115.

[23] S. S. Goncharov and V. D. Dzgoev, *Autostability of models*. ***Algebra and Logic***, vol. 19 (1980), no. 1, pp. 28–37.

[24] S. Grigorieff, *Every recursive linear ordering has a copy in DTIME-SPACE $(n, \log(n))$*. ***Journal of Symbolic Logic***, vol. 55 (1990), no. 1, pp. 260–276.

[25] I. Kalimullin and A. Melnikov, *Punctual categoricity relative to a computable oracle*. ***Lobachevskii Journal of Mathematics***, vol. 42 (2021), no. 4, pp. 735–742.

[26] I. Kalimullin, A. Melnikov, and K. M. Ng, *Algebraic structures computable without delay*. ***Theoretical Computer Science***, vol. 674 (2017), pp. 73–98.

[27] I. S. Kalimullin and A. A. Kurmacheva, *Punctual categoricity and a jump operation in the primitive recursive degrees*. ***Mathematics and Theoretical Computer Science***, vol. 2 (2024), no. 2, pp. 47–69.

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230

[28] I. S. Kalimullin, A. G. Melnikov, and K. M. Ng, *The diversity of categoricity without delay*. ***Algebra and Logic***, vol. 56 (2017), no. 2, pp. 171–177.

[29] A. I. Mal'tsev, *Constructive algebras I*. ***Russian Mathematical Surveys***, vol. 16 (1961), no. 3, pp. 77–129.

[30] C. F. D. McCoy, Δ^0_2*-categoricity of Boolean algebras and linear orderings*. ***Annals of Pure and Applied Logic***, vol. 119 (2003), nos. 1–3, pp. 85–120.

[31] M. O. Rabin, *Computable algebra, general theory and theory of computable fields*. ***Transactions of the American Mathematical Society***, vol. 95 (1960), no. 2, pp. 341–360.

[32] J. B. Remmel, *Recursive isomorphism types of recursive Boolean algebras*. ***Journal of Symbolic Logic***, vol. 46 (1981), no. 3, pp. 572–594.

[33] ———, *Recursively categorical linear orderings*. ***Proceedings of the American Mathematical Society***, vol. 83 (1981), no. 2, pp. 387–391.

DEPARTMENT OF MATHEMATICS AND MECHANICS
NOVOSIBIRSK STATE UNIVERSITY
RUSSIAN
and
DEPARTMENT OF MATHEMATICS
NAZARBAYEV UNIVERSITY
KAZAKHSTAN
E-mail: nickbazh@yandex.ru

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING (SCHOOL OF CYBER SECURITY)
UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA
CHINA
E-mail: heertern001@e.ntu.edu.sg

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE
E-mail: kmng@ntu.edu.sg

Downloaded from https://www.cambridge.org/core. UNIVERSIDAD AUTONOMA METROPOLITANA, on 16 Nov 2026 at 01:58:33, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/jsl.2026.10230