Computable Categoricity Relative to a Degree

Ph.D. General Exam

Java Darleen Villano

November 14th, 2022

University of Connecticut

Outline

1. Preliminaries
2. Overview of the exploration of computable categoricity
3. Current work

Preliminaries

Definitions

Definition

A function $f : \mathbb{N} \to \mathbb{N}$ is a (partial) **computable** function if there exists an algorithm which computes the value of f on a given input.

Definitions

Definition

A function $f : \mathbb{N} \to \mathbb{N}$ is a (partial) **computable** function if there exists an algorithm which computes the value of f on a given input.

Definition

A set $A \subseteq \mathbb{N}$ is **computable** if its characteristic function χ_A is a computable function.

Definitions

Definition

A function $f : \mathbb{N} \to \mathbb{N}$ is a (partial) **computable** function if there exists an algorithm which computes the value of f on a given input.

Definition

A set $A \subseteq \mathbb{N}$ is **computable** if its characteristic function χ_A is a computable function.

Definition

A set A is **computably enumerable** (abbreviated as **c.e.**) if it is the range of a total computable function f.

Definitions

There is an effective list $\{\Phi_e : e \in \omega\}$ of all partial computable functions in which each function is assigned a numerical index.

Definitions

There is an effective list $\{\Phi_e : e \in \omega\}$ of all partial computable functions in which each function is assigned a numerical index.

We can extend computations by allowing ourselves access to information given by a specified set, which is usually called the **oracle**.

Definitions

There is an effective list $\{\Phi_e : e \in \omega\}$ of all partial computable functions in which each function is assigned a numerical index.

We can extend computations by allowing ourselves access to information given by a specified set, which is usually called the **oracle**.

Notation

We write $f^A(n)\downarrow = a$ if f with oracle set A converges on input n and outputs a. Otherwise, we write $f^A(n)\uparrow$.

Definitions

There is an effective list $\{\Phi_e : e \in \omega\}$ of all partial computable functions in which each function is assigned a numerical index.

We can extend computations by allowing ourselves access to information given by a specified set, which is usually called the **oracle**.

Notation

We write $f^A(n) \downarrow= a$ if f with oracle set A converges on input n and outputs a. Otherwise, we write $f^A(n) \uparrow$.

Remark

If $f^A(n) \downarrow= a$, then we only use a finite initial segment σ of the oracle A to converge on n. That is, if $f^A(n) \downarrow= a$, then there exists a finite string $\sigma \subseteq A$ such that $f^\sigma(n) \downarrow= a$.

Definitions

Definition

A graph $\mathcal{G} = (G, E)$ is **computable** if its domain, G, is ω and the edge relation E is a computable relation.

Definition

A graph $\mathcal{G} = (G, E)$ is **computable** if its domain, G, is ω and the edge relation E is a computable relation.

Recall that for a graph $\mathcal{G} = (G, E)$, the edge relation E is the set

$$\{(a, b) : a, b \in G \text{ and there is an edge connecting } a \text{ and } b\}.$$

Overview of the exploration of computable categoricity

Computable categoricity

Definition

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

Computable categoricity

Definition

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

Throughout the talk, I will abbreviate "being computably categorical" to **being c.c.**

Computable categoricity

Definition

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

Throughout the talk, I will abbreviate "being computably categorical" to **being c.c.**

Example

Let $L = (A, <_L)$ be a computable linear ordering. Two elements $a, b \in A$ are said to be **adjacent** if $a <_L b$ and there is no $c \in A$ such that $a <_L c <_L b$.

L is c.c. if and only if it has only finitely many pairs of adjacent elements.

Relativizing c.c.-ness

The following relativization of c.c.-ness has been the studied extensively in the past.

Relativizing c.c.-ness

The following relativization of c.c.-ness has been the studied extensively in the past.

Definition

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

Relativizing c.c.-ness

The following relativization of c.c.-ness has been the studied extensively in the past.

Definition

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

Remark

If a structure is relatively computably categorical (abbreviated as **relatively c.c.***), then it is c.c. already.*

Relativizing c.c.-ness

The following relativization of c.c.-ness has been the studied extensively in the past.

Definition

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

Remark

If a structure is relatively computably categorical (abbreviated as **relatively c.c.***), then it is c.c. already.*

Historically, there have been two approaches in exploring the connection between c.c.-ness and relatively c.c.-ness:

Relativizing c.c.-ness

The following relativization of c.c.-ness has been the studied extensively in the past.

Definition

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

Remark

If a structure is relatively computably categorical (abbreviated as **relatively c.c.***), then it is c.c. already.*

Historically, there have been two approaches in exploring the connection between c.c.-ness and relatively c.c.-ness: an **algebraic** perspective and a **model theoretic** perspective.

The algebraic perspective

Question: Can we give algebraic characterizations of c.c.-ness in natural classes of structures?

The algebraic perspective

Question: Can we give algebraic characterizations of c.c.-ness in natural classes of structures? Are there structures where this is not possible?

Question: Can we give algebraic characterizations of c.c.-ness in natural classes of structures? Are there structures where this is not possible?

Example

- Remmel [7] showed that a computable linear ordering L is c.c. if and only if L has only finitely many pairs of adjacent elements.

The algebraic perspective

Question: Can we give algebraic characterizations of c.c.-ness in natural classes of structures? Are there structures where this is not possible?

Example

- Remmel [7] showed that a computable linear ordering L is c.c. if and only if L has only finitely many pairs of adjacent elements.
- Ershov [4] showed that an algebraically closed field is c.c. if and only if it has a finite transcendence degree over its prime subfield.

The algebraic perspective

Question: Can we give algebraic characterizations of c.c.-ness in natural classes of structures? Are there structures where this is not possible?

Example

- Remmel [7] showed that a computable linear ordering L is c.c. if and only if L has only finitely many pairs of adjacent elements.
- Ershov [4] showed that an algebraically closed field is c.c. if and only if it has a finite transcendence degree over its prime subfield.
- Goncharov, Lempp, and Solomon [5] showed that an ordered abelian group is c.c. if and only if it has finite rank.

Algebraic characterizations under relativization

Typically, if there is an algebraic characterization for being c.c. in a class of structures, then being relatively c.c. is equivalent to being c.c. for those structures.

Algebraic characterizations under relativization

Typically, if there is an algebraic characterization for being c.c. in a class of structures, then being relatively c.c. is equivalent to being c.c. for those structures.

For example, suppose a computable linear order L has finitely many adjacent pairs, and L' is any copy of L.

Algebraic characterizations under relativization

Typically, if there is an algebraic characterization for being c.c. in a class of structures, then being relatively c.c. is equivalent to being c.c. for those structures.

For example, suppose a computable linear order L has finitely many adjacent pairs, and L' is any copy of L.

We can build an isomorphism by nonuniformly matching the finitely many adjacent pairs correctly, and then extending the map by a back-and-forth construction.

Algebraic characterizations under relativization

Typically, if there is an algebraic characterization for being c.c. in a class of structures, then being relatively c.c. is equivalent to being c.c. for those structures.

For example, suppose a computable linear order L has finitely many adjacent pairs, and L' is any copy of L.

We can build an isomorphism by nonuniformly matching the finitely many adjacent pairs correctly, and then extending the map by a back-and-forth construction.

To do the back-and-forth construction, we only need to be able to compute $\leq_L$ and $\leq_{L'}$, and so the isomorphism will be computable in L'.

The algebraic perspective

There are structures which do not have a purely algebraic characterization of c.c.-ness.

The algebraic perspective

There are structures which do not have a purely algebraic characterization of c.c.-ness.

For example: partially ordered sets and rings.

The algebraic perspective

There are structures which do not have a purely algebraic characterization of c.c.-ness.

For example: partially ordered sets and rings.

We now switch to the model theoretic perspective, which will help us fill in this gap later.

The model theoretic perspective

Question: Can we use the notion of being c.c. to effectivize model theory?

Question: Can we use the notion of being c.c. to effectivize model theory?

Definition

A **Scott set** for a countable structure $\mathcal{A}$ is a set of formulas F with a fixed finite set of parameters satisfying:

(1) For each tuples $\overline{a} \in \mathcal{A}$, there is a $\varphi \in F$ such that $\mathcal{A} \models \varphi(\overline{a})$, and

(2) if $\overline{a}$ and $\overline{b}$ satisfy the same formula in F, then they are automorphic.

F is **formally** Σ_1^0 if F is a c.e. set of Σ_1^0 formulas.

The model theoretic perspective

A structure being relatively c.c. coincides with the following syntactic condition.

A structure being relatively c.c. coincides with the following syntactic condition.

Theorem (Ash, Knight, Manasse, and Slaman [1])

A structure is relatively c.c. if and only if it has a formally Σ_1 Scott family.

A structure being relatively c.c. coincides with the following syntactic condition.

Theorem (Ash, Knight, Manasse, and Slaman [1])

A structure is relatively c.c. if and only if it has a formally Σ_1 Scott family.

However, there is no syntactic characterization like the result above for *just* computable categoricity, since the index set of all c.c. structures is Π_1^1 complete [3].

The model theoretic perspective

We now pivot to a new question for the rest of this overview of the model theoretic perspective.

We now pivot to a new question for the rest of this overview of the model theoretic perspective.

Question: For structures where being c.c. cannot be algebraically characterized, what additional conditions are needed to show that being c.c. is equivalent to being relatively c.c.?

The model theoretic perspective

We now pivot to a new question for the rest of this overview of the model theoretic perspective.

Question: For structures where being c.c. cannot be algebraically characterized, what additional conditions are needed to show that being c.c. is equivalent to being relatively c.c.?

Theorem (Goncharov [6])

If a structure is c.c. and its $\forall\exists$ theory is decidable, then it is relatively c.c.

Exploring the gap

Now that we have explored some results about c.c.-ness and relatively c.c.-ness, let's take the time to motivate my current research project.

Exploring the gap

Now that we have explored some results about c.c.-ness and relatively c.c.-ness, let's take the time to motivate my current research project.

Definition

Let $\mathcal{A}$ be a computable structure. $\mathcal{A}$ is **computably categorical relative to a degree d** if for every **d**-computable copy $\mathcal{B}$ of $\mathcal{A}$, there exists a **d**-computable isomorphism between $\mathcal{A}$ and $\mathcal{B}$.

Exploring the gap

Now that we have explored some results about c.c.-ness and relatively c.c.-ness, let's take the time to motivate my current research project.

Definition

Let $\mathcal{A}$ be a computable structure. $\mathcal{A}$ is **computably categorical relative to a degree d** if for every **d**-computable copy $\mathcal{B}$ of $\mathcal{A}$, there exists a **d**-computable isomorphism between $\mathcal{A}$ and $\mathcal{B}$.

Fact

A computable structure $\mathcal{A}$ is **relatively computably categorical** *if for all $X \in 2^{\mathbb{N}}$, $\mathcal{A}$ is c.c. relative to X.*

Exploring the gap

Now that we have explored some results about c.c.-ness and relatively c.c.-ness, let's take the time to motivate my current research project.

Definition

Let $\mathcal{A}$ be a computable structure. $\mathcal{A}$ is **computably categorical relative to a degree d** if for every **d**-computable copy $\mathcal{B}$ of $\mathcal{A}$, there exists a **d**-computable isomorphism between $\mathcal{A}$ and $\mathcal{B}$.

Fact

A computable structure $\mathcal{A}$ is **relatively computably categorical** *if for all $X \in 2^{\mathbb{N}}$, $\mathcal{A}$ is c.c. relative to X.*

Question: Are there structures which are in between being c.c. and being relatively c.c.? What do they look like?

We have the following fact.

We have the following fact.

Fact ([2])

If $\mathcal{A}$ is a computable structure and it is computably categorical relative to some degree $\mathbf{d} \geq \mathbf{0}''$ (abbreviated as c.c. relative to $\mathbf{d}$), then $\mathcal{A}$ has a $\mathbf{0}''$-computable Σ_1^0 Scott family.

Exploring the gap

We have the following fact.

Fact ([2])

If $\mathcal{A}$ is a computable structure and it is computably categorical relative to some degree $\mathbf{d} \geq \mathbf{0}''$ (abbreviated as c.c. relative to $\mathbf{d}$), then $\mathcal{A}$ has a $\mathbf{0}''$-computable Σ^0_1 Scott family.

This implies that $\mathcal{A}$, as in the statement of the Fact, must be c.c. relative to all degrees above $\mathbf{0}''$.

The cone above $\mathbf{0}''$

If $\mathcal{A}$ is c.c. relative to some $\mathbf{d} \geq \mathbf{0}''$, then it is c.c. relative to *all* degrees above $\mathbf{0}''$.

The cone above $\mathbf{0}''$

If $\mathcal{A}$ is c.c. relative to some $\mathbf{d} \geq \mathbf{0}''$, then it is c.c. relative to *all* degrees above $\mathbf{0}''$.

The contrapositive of the Fact also gives us that if $\mathcal{A}$ does not have a $\mathbf{0}''$-computable Σ^0_1 Scott family, then it is not c.c. relative to *any* $\mathbf{d} \geq \mathbf{0}''$.

The cone above $\mathbf{0}''$

If $\mathcal{A}$ is c.c. relative to some $\mathbf{d} \geq \mathbf{0}''$, then it is c.c. relative to *all* degrees above $\mathbf{0}''$.

The contrapositive of the Fact also gives us that if $\mathcal{A}$ does not have a $\mathbf{0}''$-computable Σ^0_1 Scott family, then it is not c.c. relative to *any* $\mathbf{d} \geq \mathbf{0}''$.

So at $\mathbf{0}''$ and above, any computable structure $\mathcal{A}$ will settle on whether it is c.c. relative to all degrees or to none of them.

The cone above $\mathbf{0}''$

If $\mathcal{A}$ is c.c. relative to some $\mathbf{d} \geq \mathbf{0}''$, then it is c.c. relative to *all* degrees above $\mathbf{0}''$.

The contrapositive of the Fact also gives us that if $\mathcal{A}$ does not have a $\mathbf{0}''$-computable Σ_1^0 Scott family, then it is not c.c. relative to *any* $\mathbf{d} \geq \mathbf{0}''$.

So at $\mathbf{0}''$ and above, any computable structure $\mathcal{A}$ will settle on whether it is c.c. relative to all degrees or to none of them.

Question: What happens between $\mathbf{0}$ and $\mathbf{0}''$?

A surprising result

Downey, Harrison-Trainor, and Melnikov recently showed the following surprising result:

A surprising result

Downey, Harrison-Trainor, and Melnikov recently showed the following surprising result:

Theorem (Downey, Harrison-Trainor, Melnikov [2])

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

A surprising result

Downey, Harrison-Trainor, and Melnikov recently showed the following surprising result:

Theorem (Downey, Harrison-Trainor, Melnikov [2])

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

(1) $\mathcal{A}$ *is computably categorical relative to* Y_i *for each* i,

A surprising result

Downey, Harrison-Trainor, and Melnikov recently showed the following surprising result:

Theorem (Downey, Harrison-Trainor, Melnikov [2])

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

(1) *$\mathcal{A}$ is computably categorical relative to Y_i for each i,*

(2) *$\mathcal{A}$ is not computably categorical relative to X_i for each i,*

A surprising result

Downey, Harrison-Trainor, and Melnikov recently showed the following surprising result:

Theorem (Downey, Harrison-Trainor, Melnikov [2])

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

(1) *$\mathcal{A}$ is computably categorical relative to Y_i for each i,*

(2) *$\mathcal{A}$ is not computably categorical relative to X_i for each i,*

(3) *$\mathcal{A}$ is computably categorical relative to $\mathbf{0}'$.*

A surprising result

Downey, Harrison-Trainor, and Melnikov recently showed the following surprising result:

Theorem (Downey, Harrison-Trainor, Melnikov [2])

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

(1) *$\mathcal{A}$ is computably categorical relative to Y_i for each i,*

(2) *$\mathcal{A}$ is not computably categorical relative to X_i for each i,*

(3) *$\mathcal{A}$ is computably categorical relative to $\mathbf{0}'$.*

My current project is to find similar patterns in which we allow the degrees to be incomparable.

Current work

A special case

Theorem (Downey, Harrison-Trainor, Melnikov [2])

There exists a computable directed graph $\mathcal{G}$ and a c.e. set X such that

(1) *$\mathcal{G}$ is computably categorical, and*

(2) *$\mathcal{G}$ is not computably categorical relative to X.*

A special case

Theorem (Downey, Harrison-Trainor, Melnikov [2])

There exists a computable directed graph $\mathcal{G}$ and a c.e. set X such that

(1) *$\mathcal{G}$ is computably categorical, and*

(2) *$\mathcal{G}$ is not computably categorical relative to X.*

Proof sketch.

To prove this, we want to build the following:

(1) the computable directed graph $\mathcal{G}$,

A special case

Theorem (Downey, Harrison-Trainor, Melnikov [2])

There exists a computable directed graph $\mathcal{G}$ and a c.e. set X such that

(1) *$\mathcal{G}$ is computably categorical, and*

(2) *$\mathcal{G}$ is not computably categorical relative to X.*

Proof sketch.

To prove this, we want to build the following:

(1) the computable directed graph $\mathcal{G}$,

(2) the c.e. set X,

A special case

Theorem (Downey, Harrison-Trainor, Melnikov [2])

There exists a computable directed graph $\mathcal{G}$ and a c.e. set X such that

(1) *$\mathcal{G}$ is computably categorical, and*

(2) *$\mathcal{G}$ is not computably categorical relative to X.*

Proof sketch.

To prove this, we want to build the following:

(1) the computable directed graph $\mathcal{G}$,

(2) the c.e. set X,

(3) the graph $\mathcal{B}$ such that $\mathcal{B} \leq_T X$ and $\mathcal{B}$ is a copy of $\mathcal{G}$, and

A special case

Theorem (Downey, Harrison-Trainor, Melnikov [2])

There exists a computable directed graph $\mathcal{G}$ and a c.e. set X such that

(1) *$\mathcal{G}$ is computably categorical, and*

(2) *$\mathcal{G}$ is not computably categorical relative to X.*

Proof sketch.

To prove this, we want to build the following:

(1) the computable directed graph $\mathcal{G}$,

(2) the c.e. set X,

(3) the graph $\mathcal{B}$ such that $\mathcal{B} \leq_{\mathrm{T}} X$ and $\mathcal{B}$ is a copy of $\mathcal{G}$, and

(4) for each $i \in \mathbb{N}$ such that $\mathcal{M}_i \cong \mathcal{G}$, a computable isomorphism $f_i : \mathcal{G} \to \mathcal{M}_i$ (where $\mathcal{M}_i$ is the ith computable graph)

Listing our requirements

We have the following requirements:

Listing our requirements

We have the following requirements:

- S_i : if $\mathcal{G} \cong \mathcal{M}_i$, then there exists a computable isomorphism $f_i : \mathcal{G} \to \mathcal{M}_i$, and

Listing our requirements

We have the following requirements:

- S_i : if $\mathcal{G} \cong \mathcal{M}_i$, then there exists a computable isomorphism $f_i : \mathcal{G} \to \mathcal{M}_i$, and
- R_e : $\Phi_e^X : \mathcal{G} \to \mathcal{B}$ is not an isomorphism.

Listing our requirements

We have the following requirements:

- S_i : if $\mathcal{G} \cong \mathcal{M}_i$, then there exists a computable isomorphism $f_i : \mathcal{G} \to \mathcal{M}_i$, and
- R_e : $\Phi_e^X : \mathcal{G} \to \mathcal{B}$ is not an isomorphism.

We want to meet all such requirements for all $i, e \in \omega$ with our construction.

Listing our requirements

We have the following requirements:

- S_i : if $\mathcal{G} \cong \mathcal{M}_i$, then there exists a computable isomorphism $f_i : \mathcal{G} \to \mathcal{M}_i$, and
- R_e : $\Phi_e^X : \mathcal{G} \to \mathcal{B}$ is not an isomorphism.

We want to meet all such requirements for all $i, e \in \omega$ with our construction.

The S_i requirements are working towards making $\mathcal{G}$ computably categorical.

Listing our requirements

We have the following requirements:

- S_i : if $\mathcal{G} \cong \mathcal{M}_i$, then there exists a computable isomorphism $f_i : \mathcal{G} \to \mathcal{M}_i$, and
- R_e : $\Phi_e^X : \mathcal{G} \to \mathcal{B}$ is not an isomorphism.

We want to meet all such requirements for all $i, e \in \omega$ with our construction.

The S_i requirements are working towards making $\mathcal{G}$ computably categorical.

The R_e requirements are working towards making $\mathcal{G}$ *not* computably categorical relative to our c.e. set X.

Building our graphs $\mathcal{G}$ and $\mathcal{B}$

The graphs $\mathcal{G}$ and $\mathcal{B}$ will be built globally.

Building our graphs $\mathcal{G}$ and $\mathcal{B}$

The graphs $\mathcal{G}$ and $\mathcal{B}$ will be built globally.

At stage $s = 0$, we set their domains $G = B = \emptyset$.

Building our graphs $\mathcal{G}$ and $\mathcal{B}$

The graphs $\mathcal{G}$ and $\mathcal{B}$ will be built globally.

At stage $s = 0$, we set their domains $G = B = \emptyset$.

At stage $s > 0$, we add a_{2s} and a_{2s+1} as root nodes to $\mathcal{G}$ and attach 2-loop to each node. Then, we attach a $(5s+1)$-loop to a_{2s} and a $(5s+2)$-loop to a_{2s+1}.

Building our graphs $\mathcal{G}$ and $\mathcal{B}$

The graphs $\mathcal{G}$ and $\mathcal{B}$ will be built globally.

At stage $s = 0$, we set their domains $G = B = \emptyset$.

At stage $s > 0$, we add a_{2s} and a_{2s+1} as root nodes to $\mathcal{G}$ and attach 2-loop to each node. Then, we attach a $(5s + 1)$-loop to a_{2s} and a $(5s + 2)$-loop to a_{2s+1}.

We mirror these actions for our graph $\mathcal{B}$, where b_{2s} and b_{2s+1} are the corresponding root nodes.

Building our graphs $\mathcal{G}$ and $\mathcal{B}$

The graphs $\mathcal{G}$ and $\mathcal{B}$ will be built globally.

At stage $s = 0$, we set their domains $G = B = \emptyset$.

At stage $s > 0$, we add a_{2s} and a_{2s+1} as root nodes to $\mathcal{G}$ and attach 2-loop to each node. Then, we attach a $(5s+1)$-loop to a_{2s} and a $(5s+2)$-loop to a_{2s+1}.

We mirror these actions for our graph $\mathcal{B}$, where b_{2s} and b_{2s+1} are the corresponding root nodes.

Definition

The root node a_{2s} in our graph $\mathcal{G}$ with its loops is the 2s**th connected component** or just the 2sth component of $\mathcal{G}$.

Thinking of strategies

For all $i, e \in \omega$, we want to meet the requirements:

- S_i : if $\mathcal{G} \cong \mathcal{M}_i$, then there exists a computable embedding $f_i : \mathcal{G} \to \mathcal{M}_i$, and
- R_e : $\Phi_e^X : \mathcal{G} \to \mathcal{B}$ is not an isomorphism.

Thinking of strategies

For all $i, e \in \omega$, we want to meet the requirements:

- S_i : if $\mathcal{G} \cong \mathcal{M}_i$, then there exists a computable embedding $f_i : \mathcal{G} \to \mathcal{M}_i$, and
- R_e : $\Phi_e^X : \mathcal{G} \to \mathcal{B}$ is not an isomorphism.

For each type of requirement, we can think of a basic strategy to satisfy them.

Basic S_i-strategy

Fix an S_i requirement. Our strategy is:

1. Set its parameter $n_i = 0$. This parameter keeps track of the components we are currently trying to match between $\mathcal{G}$ and $\mathcal{M}_i$.

Basic S_i-strategy

Fix an S_i requirement. Our strategy is:

1. Set its parameter $n_i = 0$. This parameter keeps track of the components we are currently trying to match between $\mathcal{G}$ and $\mathcal{M}_i$. Set f_i to be the empty map.

Basic S_i-strategy

Fix an S_i requirement. Our strategy is:

1. Set its parameter $n_i = 0$. This parameter keeps track of the components we are currently trying to match between $\mathcal{G}$ and $\mathcal{M}_i$. Set f_i to be the empty map.
2. At stage s, check if $\mathcal{M}_i[s]$ contains a copy of the $2n_i$th and $(2n_i + 1)$st components of $\mathcal{G}$.

Basic S_i-strategy

Fix an S_i requirement. Our strategy is:

1. Set its parameter $n_i = 0$. This parameter keeps track of the components we are currently trying to match between $\mathcal{G}$ and $\mathcal{M}_i$. Set f_i to be the empty map.
2. At stage s, check if $\mathcal{M}_i[s]$ contains a copy of the $2n_i$th and $(2n_i + 1)$st components of $\mathcal{G}$.
    - If so, extend f_i to those components and increment n_i.

Basic S_i-strategy

Fix an S_i requirement. Our strategy is:

1. Set its parameter $n_i = 0$. This parameter keeps track of the components we are currently trying to match between $\mathcal{G}$ and $\mathcal{M}_i$. Set f_i to be the empty map.
2. At stage s, check if $\mathcal{M}_i[s]$ contains a copy of the $2n_i$th and $(2n_i + 1)$st components of $\mathcal{G}$.
    - If so, extend f_i to those components and increment n_i.
    - If not, continue the search at the next stage.

Basic R_e-strategy

Fix an R_e requirement. Our strategy is:

1. Set its parameter n_e to be large. This parameter indicates which components R_e will use to diagonalize.

Basic R_e-strategy

Fix an R_e requirement. Our strategy is:

1. Set its parameter n_e to be large. This parameter indicates which components R_e will use to diagonalize.
2. At stage s, check if $\Phi_e^X[s]$ maps the $2n_e$th and $(2n_e + 1)$st components in $\mathcal{G}$ to the corresponding components in $\mathcal{B}$.

Basic R_e-strategy

Fix an R_e requirement. Our strategy is:

1. Set its parameter n_e to be large. This parameter indicates which components R_e will use to diagonalize.
2. At stage s, check if $\Phi_e^X[s]$ maps the $2n_e$th and $(2n_e + 1)$st components in $\mathcal{G}$ to the corresponding components in $\mathcal{B}$.
   - If not, go to the next stage.

Basic R_e-strategy

Fix an R_e requirement. Our strategy is:

1. Set its parameter n_e to be large. This parameter indicates which components R_e will use to diagonalize.
2. At stage s, check if $\Phi_e^X[s]$ maps the $2n_e$th and $(2n_e + 1)$st components in $\mathcal{G}$ to the corresponding components in $\mathcal{B}$.
   - If not, go to the next stage.
   - If so, then add loops as follows:
     (a) $(5n_e + 2)$-loop and $(5n_e + 3)$-loop to the $2n_e$th component in $\mathcal{G}$

Basic R_e-strategy

Fix an R_e requirement. Our strategy is:

1. Set its parameter n_e to be large. This parameter indicates which components R_e will use to diagonalize.
2. At stage s, check if $\Phi_e^X[s]$ maps the $2n_e$th and $(2n_e + 1)$st components in $\mathcal{G}$ to the corresponding components in $\mathcal{B}$.
   - If not, go to the next stage.
   - If so, then add loops as follows:
     (a) $(5n_e + 2)$-loop and $(5n_e + 3)$-loop to the $2n_e$th component in $\mathcal{G}$
     (b) $(5n_e + 1)$-loop and $(5n_e + 4)$-loop to the $(2n_e + 1)$st component in $\mathcal{G}$

Basic R_e-strategy

Fix an R_e requirement. Our strategy is:

1. Set its parameter n_e to be large. This parameter indicates which components R_e will use to diagonalize.
2. At stage s, check if $\Phi_e^X[s]$ maps the $2n_e$th and $(2n_e+1)$st components in $\mathcal{G}$ to the corresponding components in $\mathcal{B}$.
   - If not, go to the next stage.
   - If so, then add loops as follows:
     (a) $(5n_e+2)$-loop and $(5n_e+3)$-loop to the $2n_e$th component in $\mathcal{G}$
     (b) $(5n_e+1)$-loop and $(5n_e+4)$-loop to the $(2n_e+1)$st component in $\mathcal{G}$
     (c) $(5n_e+2)$-loop and $(5n_e+4)$-loop to the $2n_e$th component in $\mathcal{B}$

Basic R_e-strategy

Fix an R_e requirement. Our strategy is:

1. Set its parameter n_e to be large. This parameter indicates which components R_e will use to diagonalize.
2. At stage s, check if $\Phi_e^X[s]$ maps the $2n_e$th and $(2n_e+1)$st components in $\mathcal{G}$ to the corresponding components in $\mathcal{B}$.
    - If not, go to the next stage.
    - If so, then add loops as follows:
        (a) $(5n_e+2)$-loop and $(5n_e+3)$-loop to the $2n_e$th component in $\mathcal{G}$
        (b) $(5n_e+1)$-loop and $(5n_e+4)$-loop to the $(2n_e+1)$st component in $\mathcal{G}$
        (c) $(5n_e+2)$-loop and $(5n_e+4)$-loop to the $2n_e$th component in $\mathcal{B}$
        (d) $(5n_e+1)$-loop and $(5n_e+3)$-loop to the $(2n_e+1)$st component in $\mathcal{B}$

Conflict between strategies

Suppose S_i has already defined f_i on the $2n_e$th and $(2n_e + 1)$st components of $\mathcal{G}$ when R_e adds loops.

Conflict between strategies

Suppose S_i has already defined f_i on the $2n_e$th and $(2n_e + 1)$st components of $\mathcal{G}$ when R_e adds loops.

In the worst case scenario, $\mathcal{M}_i$ could add the new loops to make f_i wrong.

Conflict between strategies

Suppose S_i has already defined f_i on the $2n_e$th and $(2n_e + 1)$st components of $\mathcal{G}$ when R_e adds loops.

In the worst case scenario, $\mathcal{M}_i$ could add the new loops to make f_i wrong.

How do we solve this conflict when S_i has higher priority than R_e?

Updating the R_e-strategy

We will change our R_e-strategy:

Updating the R_e-strategy

We will change our R_e-strategy:

1. Set parameter n_e to be large.

Updating the R_e-strategy

We will change our R_e-strategy:

1. Set parameter n_e to be large.
2. Check if $\Phi_e^X[s]$ maps the $2n_e$th and $(2n_e + 1)$st components in $\mathcal{G}$ to corresponding components in $\mathcal{B}$.

Updating the R_e-strategy

We will change our R_e-strategy:

1. Set parameter n_e to be large.
2. Check if $\Phi_e^X[s]$ maps the $2n_e$th and $(2n_e + 1)$st components in $\mathcal{G}$ to corresponding components in $\mathcal{B}$. If not, do nothing.

Updating the R_e-strategy

We will change our R_e-strategy:

1. Set parameter n_e to be large.
2. Check if $\Phi_e^X[s]$ maps the $2n_e$th and $(2n_e+1)$st components in $\mathcal{G}$ to corresponding components in $\mathcal{B}$. If not, do nothing. If so, let $m_e =$ max use of the computations above and proceed to Step 3.

Updating the R_e-strategy

We will change our R_e-strategy:

1. Set parameter n_e to be large.
2. Check if $\Phi_e^X[s]$ maps the $2n_e$th and $(2n_e+1)$st components in $\mathcal{G}$ to corresponding components in $\mathcal{B}$. If not, do nothing. If so, let $m_e =$ max use of the computations above and proceed to Step 3.
3. Add a $(5n_e+3)$-loop to the $2n_e$th components in $\mathcal{G}$ and in $\mathcal{B}$. Add a $(5n_e+4)$-loop to the $(2n_e+1)$st components in $\mathcal{G}$ and in $\mathcal{B}$. Make the use u_e of adding these new loops satisfy $u_e > m_e$.

Updating the R_e-strategy

4. Pause the R_e-strategy and challenge S_i to extend its map f_i to these new loops.

Updating the R_e-strategy

4. Pause the R_e-strategy and challenge S_i to extend its map f_i to these new loops.

   While waiting for S_i to meet this challenge, we start a new version of R_e that works on a pair of components on which f_i has not been defined yet.

Updating the R_e-strategy

4. Pause the R_e-strategy and challenge S_i to extend its map f_i to these new loops.

   While waiting for S_i to meet this challenge, we start a new version of R_e that works on a pair of components on which f_i has not been defined yet.

   If S_i never meets the challenge, then the new R_e-strategy can win with no interference from S_i.

Updating the R_e-strategy

4. Pause the R_e-strategy and challenge S_i to extend its map f_i to these new loops.

   While waiting for S_i to meet this challenge, we start a new version of R_e that works on a pair of components on which f_i has not been defined yet.

   If S_i never meets the challenge, then the new R_e-strategy can win with no interference from S_i. If S_i meets the challenge, then we return to the old R_e-strategy.

Updating the R_e-strategy

4. Pause the R_e-strategy and challenge S_i to extend its map f_i to these new loops.

   While waiting for S_i to meet this challenge, we start a new version of R_e that works on a pair of components on which f_i has not been defined yet.

   If S_i never meets the challenge, then the new R_e-strategy can win with no interference from S_i. If S_i meets the challenge, then we return to the old R_e-strategy.
5. If S_i meets the challenge, then R_e adds a $(5n_e + 2)$-loop to the $2n_e$th components in $\mathcal{G}$ and in $\mathcal{B}$.

Updating the R_e-strategy

4. Pause the R_e-strategy and challenge S_i to extend its map f_i to these new loops.

   While waiting for S_i to meet this challenge, we start a new version of R_e that works on a pair of components on which f_i has not been defined yet.

   If S_i never meets the challenge, then the new R_e-strategy can win with no interference from S_i. If S_i meets the challenge, then we return to the old R_e-strategy.
5. If S_i meets the challenge, then R_e adds a $(5n_e + 2)$-loop to the $2n_e$th components in $\mathcal{G}$ and in $\mathcal{B}$. It adds a $(5n_e + 1)$-loop to the $(2n_e + 1)$st components in $\mathcal{G}$ and in $\mathcal{B}$.

Updating the R_e-strategy

4. Pause the R_e-strategy and challenge S_i to extend its map f_i to these new loops.

   While waiting for S_i to meet this challenge, we start a new version of R_e that works on a pair of components on which f_i has not been defined yet.

   If S_i never meets the challenge, then the new R_e-strategy can win with no interference from S_i. If S_i meets the challenge, then we return to the old R_e-strategy.
5. If S_i meets the challenge, then R_e adds a $(5n_e + 2)$-loop to the $2n_e$th components in $\mathcal{G}$ and in $\mathcal{B}$. It adds a $(5n_e + 1)$-loop to the $(2n_e + 1)$st components in $\mathcal{G}$ and in $\mathcal{B}$. Finally, enumerate u_e into X, and this lets us swap the $(5n_e + 3)$ and $(5n_e + 4)$-loops in $\mathcal{B}$.

Looking at minimal pairs

We can extend my construction to build a **minimal pair** of sets such that a computable graph $\mathcal{G}$ is not c.c. relative to either set.

Looking at minimal pairs

We can extend my construction to build a **minimal pair** of sets such that a computable graph $\mathcal{G}$ is not c.c. relative to either set.

Theorem (Villano; Minimal Pairs Version)

There exists a computable directed graph $\mathcal{G}$ and c.e. sets A_0 and A_1 such that

(1) *$\mathcal{G}$ is computably categorical,*

(2) *$\mathcal{G}$ is not computably categorical relative to A_i for $i = 0, 1$, and*

(3) *A_0 and A_1 form a minimal pair.*

Looking at minimal pairs

We can extend my construction to build a **minimal pair** of sets such that a computable graph $\mathcal{G}$ is not c.c. relative to either set.

Theorem (Villano; Minimal Pairs Version)

There exists a computable directed graph $\mathcal{G}$ and c.e. sets A_0 and A_1 such that

(1) *$\mathcal{G}$ is computably categorical,*

(2) *$\mathcal{G}$ is not computably categorical relative to A_i for $i = 0, 1$, and*

(3) *A_0 and A_1 form a minimal pair.*

Question: Are other combinations possible with minimal pairs?

More structural results

Theorem (Villano)

There exists a computable directed graph $\mathcal{G}$ and a c.e. set X such that

(1) *$\mathcal{G}$ is not computably categorical, and*

(2) *$\mathcal{G}$ is computably categorical relative to X.*

More structural results

Theorem (Villano)

There exists a computable directed graph $\mathcal{G}$ and a c.e. set X such that

(1) *$\mathcal{G}$ is not computably categorical, and*

(2) *$\mathcal{G}$ is computably categorical relative to X.*

Theorem (Villano; Minimal Pairs Version (Not C.C. Root Node))

There exists a computable directed graph $\mathcal{G}$ and c.e. sets A_0 and A_1 such that

(1) *$\mathcal{G}$ is not computably categorical,*

(2) *$\mathcal{G}$ is computably categorical relative to A_i for $i = 0, 1$, and*

(3) *A_0 and A_1 form a minimal pair.*

Goals

Conjecture (Splitting the Minimal Pair (C.C. Root Node))

There exists a computable directed graph $\mathcal{G}$ and c.e. sets X and Y such that

(1) $\mathcal{G}$ is computably categorical,

(2) $\mathcal{G}$ is computably categorical relative to X,

(3) $\mathcal{G}$ is not computably categorical relative to Y, and

(4) X and Y form a minimal pair.

Goals

Conjecture (Splitting the Minimal Pair (C.C. Root Node))

There exists a computable directed graph $\mathcal{G}$ and c.e. sets X and Y such that

(1) $\mathcal{G}$ is computably categorical,

(2) $\mathcal{G}$ is computably categorical relative to X,

(3) $\mathcal{G}$ is not computably categorical relative to Y, and

(4) X and Y form a minimal pair.

We also have the version where we let $\mathcal{G}$ be not c.c.

Goals

Conjecture (Splitting the Minimal Pair (C.C. Root Node))

There exists a computable directed graph $\mathcal{G}$ and c.e. sets X and Y such that

(1) $\mathcal{G}$ is computably categorical,

(2) $\mathcal{G}$ is computably categorical relative to X,

(3) $\mathcal{G}$ is not computably categorical relative to Y, and

(4) X and Y form a minimal pair.

We also have the version where we let $\mathcal{G}$ be not c.c.

Lastly, we also want to see if it would be possible to make a graph $\mathcal{G}$ and c.e. sets X and Y such that $\mathcal{G}$ is c.c., is not c.c. relative to either X or Y, but is c.c. relative to their join $X \oplus Y$.

References

[1] Chris Ash et al. “Generic copies of countable structures”. *APAL* 42.3 (1989), pp. 195–205.

[2] Rodney Downey, Matthew Harrison-Trainor, and Alexander Melnikov. “Relativizing computable categoricity”. *PAMS* 149.9 (2021), pp. 3999–4013.

[3] Rodney G. Downey et al. “The complexity of computable categoricity”. *Advances in Mathematics* 268 (2015), pp. 423–466.

[4] Ju. L. Erš. “Theorie Der Numerierungen III”. *MLQ* 23.19-24 (1977), pp. 289–371. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/malq.19770231902`.

[5] Sergey S. Goncharov, Steffen Lempp, and Reed Solomon. "The computable dimension of ordered abelian groups". *Advances in Mathematics* 175.1 (2003), pp. 102–143.

[6] S. S. Gončarov. "The problem of the number of nonautoequivalent constructivizations". *Algebra i Logika* 19.6 (1980), pp. 621–639, 745.

[7] J. B. Remmel. "Recursively Categorical Linear Orderings". *PAMS* 83.2 (1981), pp. 387–391. (Visited on 10/10/2022).