COMPUTATIONAL COMPLEXITY OF AUTOMATIC STRUCTURES

A Dissertation

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

by

Jacob Carson

Julia Knight, Director

Graduate Program in Mathematics
Notre Dame, Indiana
July 2011
COMPUTATIONAL COMPLEXITY OF AUTOMATIC STRUCTURES

Abstract

by

Jacob Carson

This dissertation creates automatic structures with complex recursion-theoretic properties. A structure is said to be automatic if its universe and relations can be recognized by a deterministic finite automaton. Although arbitrary computable structures isomorphic to an automatic structure can be surprisingly complicated, automatic copies require only a few extra conditions to be computably isomorphic to each other.

The first section provides definitions and theorems necessary for creating the structures and proving their properties. The second section presents a family of automatic linear orderings, then defines a corresponding family of automatic relations on these structures which appear at arbitrarily high finite levels of the arithmetical hierarchy. The third section creates an automatic equivalence structure, and proves that the set of isomorphic equivalence structures is as complicated as any set of isomorphic equivalence structures can be. The fourth section defines a generalization of equivalence structures, called nested equivalence structures. It goes on to produce an automatic nested equivalence structure for which the set of isomorphic structures is more complicated than any set of isomorphic equivalence structures.

The fifth provides brief descriptions of other projects the author assisted during his time at Notre Dame, as well as suggestions for future work on related problems.
This dissertation is dedicated to

- Professor Julia Knight, who suggested the problems to me, conjectured the results of the theorems, and provided mathematical and emotional support in proving the theorems.

- Professor Douglas Cenzer, who suggested the subject to Julia Knight.

- Sara Quinn, whose previous work on equivalence structures provided a template for theorems about the nested equivalence structure, and whose dissertation provided a template for the format and style of this dissertation.

- Mom and Dad, whose love encouraged me to see this project through.
CONTENTS

Figures .......................................................... v

Chapter 1: Introduction ........................................... 1
  1.1 Background .................................................. 1
    1.1.1 Deterministic Finite Automata ......................... 1
    1.1.2 Automatic Structures ................................. 3
    1.1.3 Notation for Tuples ................................. 4
  1.2 Previous Work .............................................. 5
    1.2.1 The Arithmetical Hierarchy ........................... 5
      1.2.1.1 The Standard Back and Forth Relations ......... 5
      1.2.1.2 Categoricity .................................... 6
      1.2.1.3 Intrinsic and Relatively Intrinsic ............. 8
    1.2.2 General Theorems about Automatic Structures ....... 9
  1.3 This Dissertation ........................................ 12

Chapter 2: Automatic Ordinals ................................. 13
  2.1 Limits Within Automatic Ordinals ....................... 13
    2.1.1 Limits ............................................... 13
    2.1.2 The General Case .................................... 18
      2.1.2.1 The Back and Forth Relations for General Linear
              Orderings ........................................ 18
      2.1.2.2 The Back and Forth Relations for Ordinals .... 18
      2.1.2.3 The $J$-Limits in Ordinals ..................... 20
  2.2 Categoricity of $\omega^n$ ................................. 21

Chapter 3: Automatic Equivalence Relations ................... 24
  3.1 General Facts about Automatic Equivalence Structures .... 24
    3.1.1 Definitions and Notation ............................. 24
    3.1.2 Theorems ............................................. 25
  3.2 Constructing an Automatic Equivalence Structure .......... 26
    3.2.1 Describing the Desired Equivalence Structure ...... 26
FIGURES

1.1 A simple example of a deterministic finite automaton. The circles represent the states of the automaton, and the arrows represent the transitions. The circle with an “S” in it is the start state. The circle with the double-line is an accept state; the circle with a single line is a reject state. The “1”’s mean the transitions are taken when the machine reads a “1.” A bit of examination makes it clear that this machine accepts strings of 1’s of even length. . . . . . . . . . 2

3.1 The automaton that recognizes the equivalence relation of $A_\infty$. The symbol on the bottom-right loop is shorthand for the list of all possible symbols; once the machine enters the state with the loop with that label, it is guaranteed to accept the string it reads. . . . 27

3.2 The machine that recognizes the universe of $A_1$. . . . . . . . . 28

4.1 The automaton that recognizes the universe of $A_2$. . . . . . . . . 32

4.2 A finite automaton that recognizes $\sim_1$. . . . . . . . . . . . . . . . 34

4.3 A finite automaton that recognizes $\sim_2$. . . . . . . . . . . . . . . . 35
CHAPTER 1

INTRODUCTION

1.1 Background

Computable structure theory is a branch of logic that combines computability and model theory. It covers mathematical structures for which the universe and relations can all be calculated by a Turing Machine. This branch can be further refined by putting additional restrictions on the computations allowed in recognizing the structure.

1.1.1 Deterministic Finite Automata

Definition 1. A deterministic finite automaton is a tuple \( \langle \Sigma, S, t, A, s \rangle \) where

- \( \Sigma \) is an alphabet – a finite set of symbols that can be written down to make words
- \( S \) is a finite set of states that the automaton can take
- \( t \) is a function \( S \times \Sigma \rightarrow S \) that reads in the current state of the machine and a letter of the alphabet, and returns the next state that the automaton will enter
- \( A \) is the set of states that will accept a word if the automaton completes its reading while in that state
• $s$ is the starting state for this automaton

A finite automaton reads strings left to right one character at a time. It starts in state $s$, and transitions to its next state by applying $t$ to the current state and the current character. Once it is finished reading the string, it will accept or reject the string based on whether or not the state it finishes in is in $A$.

A finite automaton is typically visualized as a labeled directed multigraph. The vertices correspond to the states of the machine. They are labeled based on whether or not they are accepting states, and one state is labeled as being the starting state. The edges are labeled with the symbol(s) that causes the machine to transition from the state at the start of the arrow to the state at the end of the arrow.

Displaying all of a finite automaton usually results in an unintelligible mess of a diagram, so there are some conventions used to simplify the graph. If an edge is labeled by a set of characters, that is an abbreviation for the graph having one edge for each character in the set. This is useful when trying to make a transition

![Diagram of a finite automaton](image)

Figure 1.1. A simple example of a deterministic finite automaton. The circles represent the states of the automaton, and the arrows represent the transitions. The circle with an “S” in it is the start state. The circle with the double-line is an accept state; the circle with a single line is a reject state. The “1”s mean the transitions are taken when the machine reads a “1.” A bit of examination makes it clear that this machine accepts strings of 1’s of even length.
that occurs when the machine reads any character; just label an edge with the symbol for the alphabet. If a vertex does not have any edge leading out of it labeled with a character, that means the machine automatically rejects the string it reads if it enters that state and reads that character. Given a graph representing a machine that uses these abbreviations, it is straightforward to create a graph that does not use these abbreviations. However, the resulting graph will probably be very crowded.

Finite automata are computable tools for choosing words from an alphabet to be included in a language. They are weaker than Turing Machines; that is, there are languages that can be accepted by a Turing Machine that are not accepted by any finite automaton[9]. However, this weakness makes these automata easier to work with than Turing Machines. There is an algorithm that determines whether or not a finite automaton accepts any words at all, whereas there is no similar algorithm for an arbitrary Turing Machine[7].

1.1.2 Automatic Structures

**Definition 2.** A structure $\mathcal{A} = \langle A, R_1, R_2, \ldots \rangle$ is automatic iff

1. There is a deterministic finite automaton that recognizes $A$

2. For each $R_i$, there is a deterministic finite automaton that recognizes $R_i$.

When recognizing relations on two or more places, a new alphabet is constructed out of tuples of letters in the old alphabet as well as an additional symbol for blanks after the end of an input. The machine that recognizes a multiple-place relation then acts on inputs written in the new alphabet.

For example, if we consider a two-place relation on a language where the only character in the alphabet is 1, the new language would have the alphabet
\{\langle1\rangle, \langle2\rangle, \langle3\rangle\}$. The square is the additional symbol for blanks after the end of an input. If we wanted to feed the strings “111” and “11111” into a machine, the string that we would enter would thus be \langle1\rangle\langle1\rangle\langle1\rangle\langle□\rangle\langle□\rangle.

In this dissertation, the symbol \langle\Sigma\rangle is used for the set of all pairs that have a character from the unextended alphabet in the top and a 1 in the bottom. The \Sigma symbol is replaced with a □ symbol if the extended alphabet is desired. Similar symbols are used for other characters in the bottom location or top location.

There are alternate ways to deal with multiple place relations. For example, an ordinary finite automaton simply reads strings; it cannot write strings. By contrast, a deterministic finite transducer will give an output string based on the transitions it follows in reading the input string, making them well suited for defining functions. There are also Thurston automatic groups, which do not have a single automaton that recognizes all multiplication. Instead, each generator of the group has its own automaton, which accepts a pair of words if the second word is the result of multiplying the first word by the generator. Although interesting, these definitions are not the focus of this dissertation; see [6] for more information.

Automatic structures should be simpler than general computable structures; this point will be expanded on in section 1.2.2. Despite this, automatic structures can be isomorphic to structures which are more complicated. Similarly, it is possible to make simple definitions within automatic structures where the simplicity cannot be carried to other copies of the structure.

1.1.3 Notation for Tuples

**Definition 3.** Given a tuple \(\overrightarrow{x}\), let \(x_i\) refer to the \(i^{th}\) individual variable in the tuple. Let \(\{\overrightarrow{x}\}\) be the total number of variables in the tuple.
1.2 Previous Work

1.2.1 The Arithmetical Hierarchy

The definitions and theorems in this subsection are due to [2].

1.2.1.1 The Standard Back and Forth Relations

Definition 4. Let \( \mathcal{A} \) and \( \mathcal{B} \) be structures over the same language, and let \( \bar{a} \in \mathcal{A} \) and \( \bar{b} \in \mathcal{B} \) be tuples where \( \{ \bar{a} \} = \{ \bar{b} \} \).

- \( (\mathcal{A}, \bar{a}) \leq_1 (\mathcal{B}, \bar{b}) \) iff, for any finitary existential formula \( \phi \), \( \mathcal{B} \models \phi(\bar{b}) \) implies \( \mathcal{A} \models \phi(\bar{a}) \).

- \( (\mathcal{A}, \bar{a}) \leq_{n+1} (\mathcal{B}, \bar{b}) \) iff, for any \( \bar{c} \in \mathcal{A} \), there is a tuple \( \bar{d} \in \mathcal{B} \) such that \( (\mathcal{B}, \bar{b}, \bar{d}) \leq_n (\mathcal{A}, \bar{a}, \bar{c}) \).

When comparing tuples from the same structure, people often abbreviate \( (\mathcal{A}, \bar{a}) \leq_n (\mathcal{A}, \bar{b}) \) to \( \bar{a} \leq_n \bar{b} \).

The primary reason I will use these relations is explained in the following theorem of Karp.

Proposition 1.2.1. Let \( \mathcal{A} \) and \( \mathcal{B} \) be countable structures, and let \( \bar{a} \in \mathcal{A} \) and \( \bar{b} \in \mathcal{B} \) be tuples where \( \{ \bar{a} \} = \{ \bar{b} \} \). Then for any countable ordinal \( \beta \geq 1 \), the following are equivalent:

1. \( (\mathcal{A}, \bar{a}) \leq_{\beta} (\mathcal{B}, \bar{b}) \).

2. The \( \Sigma_0^\beta \) formulas true of \( \bar{b} \) in \( \mathcal{B} \) are true of \( \bar{a} \) in \( \mathcal{A} \).

3. The \( \Pi_0^\beta \) formulas true of \( \bar{a} \) in \( \mathcal{A} \) are true of \( \bar{b} \) in \( \mathcal{B} \).
1.2.1.2 Categoricity

Definition 5. A structure $\mathcal{A}$ is said to be computably categorical iff for any computable $\mathcal{B}$ isomorphic to $\mathcal{A}$, there is a computable isomorphism from $\mathcal{A}$ to $\mathcal{B}$.

Definition 6. Let $\Gamma$ be a level in the arithmetical hierarchy. A structure $\mathcal{A}$ is said to be $\Gamma$-categorical iff for any computable $\mathcal{B}$ isomorphic to $\mathcal{A}$, there is an isomorphism in $\Gamma$ from $\mathcal{A}$ to $\mathcal{B}$.

Definition 7. Let $\Gamma$ be a level in the arithmetical hierarchy. A structure $\mathcal{A}$ is said to be relatively $\Gamma$-categorical iff for any $\mathcal{B}$ isomorphic to $\mathcal{A}$, there is an isomorphism in $\Gamma$ relative to $\mathcal{B}$ from $\mathcal{A}$ to $\mathcal{B}$.

The least level of categoricity of a structure can be used as a measure of the complexity of the structure; the lower the level, the easier it is to figure out the identity of the structure under consideration. Likewise, the higher the level, the more complicated sets you can code into the structure.

Lemma 1.2.2. A computable structure is relatively $\Delta^0_0$-categorical iff it has a formally $\Sigma^0_\alpha$ Scott family.

Definition 8. Let $\alpha$ be an ordinal. A structure $\mathcal{A}$ is $\alpha$-friendly iff $\mathcal{A}$ is computable, and for $\beta < \alpha$, the back-and-forth relations $\leq_\beta$ on tuples from $\mathcal{A}$ are c.e., uniformly in $\beta$.

Definition 9. Let $\mathcal{A}$ be a computable structure, $\alpha$ an ordinal, and $\vec{a}$ and $\vec{c}$ tuples from $\mathcal{A}$. The tuple $\vec{a}'$ is $\alpha$-categorically-free over $\vec{c}$ iff for all tuples $\vec{b}$ and all $\beta < \alpha$, there exist $\vec{a}'$ and $\vec{b}'$ such that $\vec{c}, \vec{a}, \vec{b} \leq_\beta \vec{c}, \vec{a}', \vec{b}'$ but $\vec{c}, \vec{a}' \not\leq_\alpha \vec{c}, \vec{a}$.

In the greater literature, the notions of “relationally-free” and “categorically-free” are each referred to simply as “free.” Usually, the context is sufficient to
determine which notion of freeness is relevant. This dissertation adds the prefixes so as to prevent confusion.

The two notions of freeness are both related to the difficulty involved in finding tuples that satisfy relations in the structure, but which are not mapped to each other by any isomorphism. Relational-freeness involves finding tuples that look alike in the structure, but not in the new relation. Categorical-freeness involves finding tuples that look alike according to low levels of the back-and-forth relations, but not at a higher level.

**Theorem 1.2.3.** Let $\mathcal{A}$ be $\alpha$-friendly. Suppose that for each tuple $\overrightarrow{c}$ in $\mathcal{A}$, there is a tuple $\overrightarrow{a}$ that is $\alpha$-categorically-free over $\overrightarrow{c}$. Finally, suppose that the relation $\not \approx^\alpha$ is computably enumerable. Then there is a computable $\mathcal{B} \cong \mathcal{A}$ with no $\Delta^0_\alpha$ isomorphism from $\mathcal{A}$ to $\mathcal{B}$.

This theorem is a useful tool for proving that a structure is not categorical at a given level – provided we can show that the standard back-and-forth relations are equivalent to something whose complement is c.e.

**Definition 10.** A structure is rigid iff it has no non-trivial automorphisms.

**Definition 11.** Let $\mathcal{A}$ be a structure. A defining family is a set $\Phi$ of formulas such that

1. each element of $\mathcal{A}$ satisfies some formula of $\Phi$

2. no formula in $\Phi$ is satisfied by two distinct elements of $\mathcal{A}$.

**Theorem 1.2.4.** If $\mathcal{A}$ is a countable structure, then $\mathcal{A}$ is rigid iff it has a defining family consisting of $L_{\omega_1 \omega}$ formulas with no parameters.
Definition 12. Let $\mathcal{A}$ be a computable structure and $\alpha$ an ordinal. $\mathcal{A}$ is $\Delta^0_\alpha$-stable if, for every computable structure $\mathcal{B} \cong \mathcal{A}$, every isomorphism from $\mathcal{B}$ to $\mathcal{A}$ is $\Delta^0_\alpha$.

$\mathcal{A}$ is said to be strictly $\Delta^0_\alpha$-stable if it is $\Delta^0_\alpha$ stable and not $\Delta^0_\beta$-stable for any $\beta < \alpha$.

Stability is thus a stronger condition than categoricity.

1.2.1.3 Intrinsic and Relatively Intrinsic

In general computable copies of a structure, one has a large degree of freedom when defining arbitrary relations. However, when a relation has to satisfy certain sentences, the possible complexity of a relation can be bounded above.

Definition 13. Let $\mathcal{A}$ be a computable structure, $R$ a relation defined on $\mathcal{A}$, $\vec{a}$ a tuple from $\mathcal{A}$ of length equal to the arity of $R$, $\vec{c}$ any tuple from $\mathcal{A}$, and $\alpha$ an ordinal. $\vec{a}$ is $\alpha$-relationally-free over $\vec{c}$ iff

1. $\vec{a} \in R$
2. For all $\vec{a}_1$ and all $\beta < \alpha$, one can effectively find $\vec{a}' \notin R$ and $\vec{a}_1'$ such that $\vec{c}, \vec{a}, \vec{a}_1 \leq_\beta \vec{c}, \vec{a}', \vec{a}_1'$

Definition 14. Let $\mathcal{A}$ be a structure and $R$ a relation defined on $\mathcal{A}$. $R$ is intrinsically $\Sigma^0_\alpha$ iff, for any computable isomorphic copy $\mathcal{B}$ and any isomorphism $F$ from $\mathcal{A}$ to $\mathcal{B}$, the image of $R$ is $\Sigma^0_\alpha$.

Definition 15. Let $\mathcal{A}$ be a structure and $R$ a relation defined on $\mathcal{A}$. $R$ is relatively intrinsically $\Sigma^0_\alpha$ iff, for any isomorphic copy $\mathcal{B}$ and any isomorphism $F$ from $\mathcal{A}$ to $\mathcal{B}$, the image of $R$ is $\Sigma^0_\alpha$ with respect to $\mathcal{B}$.

There are similar definitions for the $\Pi$ and $\Delta$ degrees as well.
When there are two similar terms where one is unmodified and the other is “relative,” the difference between the two definitions is that only computable copies are considered in the unmodified case, whereas every copy is considered in the relative case.

**Theorem 1.2.5.** For a computable structure $\mathcal{A}$ and a relation $R$ defined on $\mathcal{A}$, $R$ is relatively intrinsically $\Sigma^0_\alpha$ iff $R$ is definable in $\mathcal{A}$ by a $\Sigma^0_\alpha$ formula.

**Theorem 1.2.6.** Let $\mathcal{A}$ be an $\alpha$-friendly structure, and $R$ a computable relation on $\mathcal{A}$. If, for each tuple $\vec{c}$ in $\mathcal{A}$, there is a tuple $\vec{a}$ that is $\alpha$-relationally-free over $\vec{c}$, then $\mathcal{A}$ has a computable copy in which the image of $R$ is not $\Sigma^0_\alpha$.

This theorem is useful for proving that a relation is not intrinsically $\Sigma^0_\alpha$, which is stronger than proving that a relation is not relatively intrinsically $\Sigma^0_\alpha$.

### 1.2.2 General Theorems about Automatic Structures

**Theorem 1.2.7** (Khoussainov, Nerode; 1995. Blumensath, Gradel; 2000). There is an algorithm that takes an automatic structure $\mathcal{A}$ and a formula $\phi(\vec{x})$ and produces an automaton that recognizes exactly those tuples $\vec{a}$ that make $\phi$ true in $\mathcal{A}$. The formula $\phi$ can be in an expanded version of first order language, with the additional quantifiers “There exists infinitely many” and “There exists $m \mod n$.”

In particular, any first-order definable relation on an automatic structure will itself be an automatic relation. As such, the first order theory of any automatic structure is decidable.

The quantifier “There exists infinitely many” will be used in this dissertation, where it is denoted $\exists^\infty$. 

9
**Theorem 1.2.8.** Let $\mathcal{A}$ be an automatic structure for which there is a c.e. set of formulas $\Phi$ with fixed parameters $\overrightarrow{c}$ and finite quantifiers, including the quantifiers $\exists^\infty$ and “There exists $m \text{ mod } n,”$ such that

- For any tuple $\overrightarrow{a}$ in $\mathcal{A}$, there is some $\phi \in \Phi$ such that $\mathcal{A} \models \phi(\overrightarrow{c}, \overrightarrow{a})$.
- If $\overrightarrow{a}$ and $\overrightarrow{b}$ are tuples satisfying the same formula $\phi \in \Phi$, then $(\mathcal{A}, \overrightarrow{c}, \overrightarrow{a}) \cong (\mathcal{B}, \overrightarrow{c}, \overrightarrow{b})$.

Then any two automatic copies of $\mathcal{A}$ are computably isomorphic.

**Proof.** Let $\mathcal{B}$ and $\mathcal{C}$ be any two automatic copies of $\mathcal{A}$. Let $\overrightarrow{d}$ be the constants from $\mathcal{B}$ required for $\Phi$, and $\overrightarrow{k}$ the constants from $\mathcal{C}$. For any $b \in \mathcal{B}$, to compute the image of $b$ in $\mathcal{C}$,

1. Order the universes of $\mathcal{B}$ and $\mathcal{C}$.

2. Start with $b_1$, and enumerate a $\phi_i \in \Phi$ such that $\mathcal{B} \models \phi_i(\overrightarrow{d}, b_1)$. Stop enumerating.

3. Use the algorithm from Theorem 1.2.7 to create the machine $M_i$ that recognizes $\phi_i$.

4. Starting with the first element of $\mathcal{C}$, look for an element that $M_i$ accepts; call it $c_{i'}$.

5. Make a tuple $\overrightarrow{c}$ out of $c_{i'}$ and $c_j$, the first unmatched element of $\mathcal{C}$.

6. Enumerate a $\phi'_{i'} \in \Phi$ such that $\mathcal{C} \models \phi'_{i'}(\overrightarrow{k}, \overrightarrow{c})$.

7. Use the algorithm from Theorem 1.2.7 to create the machine $M'_{i'}$ that recognizes $\phi_i$. 

10
8. Starting with the first element $b_{j'}$ of $B$ that has not been assigned an image, test $M'_i$ on $(\overrightarrow{d}, b_1, b_{j'})$ until a tuple is accepted; assign $b_{j'}$ the image $c_j$.

9. Concatenate the next unassigned element of $B$ to the tuple $\overrightarrow{b} = (b_1, b_{j'})$, then repeat from step 2, replacing $b_1$ with $\overrightarrow{b}$.

So, each element of $B$ is mapped to an element of $C$ that satisfies the same formulas, and no two elements of $B$ are mapped to the same element. Since $B$ and $C$ are isomorphic, a tuple will be found in step 8. 

**Theorem 1.2.9.** Let $A$ be an automatic structure, and $R$ a relation on $A$ defined by a c.e. disjunction of formulas $\phi_i$ with finite quantifiers, including the quantifiers $\exists^\infty$ and “There exists $m$ mod $n$.” Then in any automatic copy of $A$, the image of $R$ is computably enumerable.

**Proof.** To enumerate the image of $R$, first order the universe of $A$. In step $\langle i, j \rangle$, enumerate the $i$th disjunct. Use the algorithm from Theorem 1.2.7 to create a machine that recognizes the tuples that make the disjunct true. Next, test the $j$th element of $A$. If the machine accepts, enumerate the element into the image; otherwise, don’t enumerate it. Repeat. 

Thus, information in automatic copies of a structure can be picked out more readily than information in general copies of a structure.

**Theorem 1.2.10** (Blumensath, Grädel; 2000). A structure is automatic if and only if it is first-order definable in the word structure

$$\langle \{0, 1\}^*, \preceq, Left, Right, EqL \rangle$$

where
• \( \{0,1\}^* \) is the set of all strings of 0’s and 1’s, including the empty string.

• \( \preceq \) is the prefix relation; that is, \( a \preceq b \) iff \( a \) is the first part of \( b \).

• \( \text{Left} \) is a function that adds a 0 to the end of any input string.

• \( \text{Right} \) is a function that adds a 1 to the end of any input string.

• \( \text{EqL} \) is a two place relation that is true when both inputs have equal length.

This theorem provides another way to determine if a structure is automatic. For certain purposes, such as proving general results about automatic structures rather than a specific structure, this method is more convenient than the definition in terms of automata.

1.3 This Dissertation

In unpublished work, Douglas Cenzer constructed an equivalence structure that has an automatic copy, and showed that any two automatic copies must be computably isomorphic. However, he also created a computable copy that was isomorphic to the automatic copy, but not computably isomorphic. This dissertation aims to extend this work by creating further structures that are automatic, but not categorical at higher levels in the arithmetical hierarchy. In addition, this dissertation will define relations on an automatic structure which are also automatic, but which are of increasing complexity in arbitrary, non-automatic isomorphic copies.
2.1 Limits Within Automatic Ordinals

2.1.1 Limits

In [7], the automatic well-orderings were shown to be exactly those less than $\omega^\omega$. Consider the structure $\mathcal{R} = \langle \omega^N, < \rangle$, where $N$ is a finite number greater than 1, and $<$ is the usual less than relation. Define a new relation $R_1$ to be the one-place relation that picks out those elements that are limit points. $R_1$ can be defined by the formula $\rho_1(x)$, where

$$\rho_1(x) = (\exists y)(y < x) \land (\forall y)(\exists z)(y < x \Rightarrow (y < z \land z < x))$$

As such, $R_1$ is guaranteed to be computably enumerable in automatic copies of $\mathcal{R}$. Indeed, since the compliment of an automatic relation is automatic, $R_1$ is pure computable. However, since $R_1$ has a definition as a $\Pi^0_2$ formula, there is no such guarantee in the computable copies of $\mathcal{R}$.

**Theorem 2.1.1.** $R_1$ is not relatively intrinsically $\Sigma^0_2$ in $\langle \omega^N, < \rangle$ for $N > 1$.

**Proof.** Assume there is a $\Sigma^0_2$ formula $\Phi(x, \overrightarrow{c})$ that defines $R_1$, where $\overrightarrow{c}$ is some tuple of constants. Let $r$ be an element of $\omega^N$ greater than any of the constants in $\overrightarrow{c}$. We show that different values for the variables will make $\Phi$ define $r + 1$,
which is a contradiction. Being a $\Sigma_0^2$ formula, $\Phi$ will necessarily have the form
\[ \forall_i (\exists u^i) (\Phi_i(x, \bar{c}, \bar{u}^i)) \]
for some (possibly infinite) range of $i$, some tuples $\bar{u}^i$, and some disjuncts $\Phi_i$. Let $I$ be an index for any disjunct true for $r$, and $\bar{d}'$ a witness for the corresponding existential quantifier. Now consider $\Phi_I(x, \bar{c}, \bar{d}')$, which is necessarily a $\Pi_1^0$ formula. To show that $\Phi_I(r, \bar{c}, \bar{d}')$ implies $\Phi_I(r+1, \bar{c}, \bar{d}')$ for some $\bar{d}'$, note that equivalence of the second and third parts of Proposition 1.2.1 means that it is sufficient to show that the $\Sigma_1^0$ formulas true of $r + 1, \bar{c}, \bar{d}'$ are true of $r, \bar{c}, \bar{d}'$. So, let $\Psi$ be a $\Sigma_1^0$ formula true of $r + 1, \bar{c}, \bar{d}'$. As before, $\Psi$ will have the form $\forall_j (\exists v_j^y) (\Psi_j(r + 1, \bar{c}, \bar{d}', \bar{v}_j^y))$. Let $J$ be the index of a true disjunct and $\bar{e}'$ a witness for $\bar{v}_j^y$. At this point, $\Psi_j$ has no quantifiers, and can be considered a conjunction of disjunctions of atomic sentences and negations thereof. Proving the lemma simply requires proving that each possible atomic sentence over $r + 1, \bar{c}, \bar{d}', \bar{e}'$ is true of $r, \bar{c}, \bar{d}', \bar{e}'$ for the right values of $\bar{d}'$ and $\bar{e}'$.

For each $d_k$ in $\bar{d}'$, if $d_k < r$, let the corresponding $d'_k$ be $d_k$. Otherwise, let $d'_k$ be $d_k + 1$. For each $e'_h$ in $\bar{e}'$, if $e'_h = r$ then let $e_h$ be any element less than $r$ but greater than any $c_l, d_k$ or $e_i$ less than $r$. For other $e'_h < r + 1$, let the corresponding $e_h$ be $e'_h$. Otherwise, if $e'_h$ is not a limit, let $e_h$ be such that $e_h + 1 = e'_h$. If $e'_h$ is a limit, let $e_h$ be an element less than $e'_h$ but greater than $r$ and any $d'_k$ or $e'_i$ that are also less than $e'_h$ (since $e'_h$ is a limit, there will be an infinite number of elements greater than $r$ and any $d'_k$ or $e'_i$ less than $e'_h$ to choose from).

Consider the sentences of the form $x < y$.

- Sentences starting with $r + 1$

$r + 1 < c_l$ : This will never be true, since $r$ was chosen to be greater than any $c_l$.

$r + 1 < d'_k$ : This will also be true of $r < d_k$, because for $d_k > r$, $d_k + 1 = d'_k$. 

14
$r + 1 < e'_h$ : This will also be true of $r < e_h$:

* If $e'_h$ is a limit, $e_h$ is greater than $r$ because $r + 1$ is not a limit.
* If $e'_h$ is not a limit, $e_h + 1 = e_h$.

- Sentences starting with an element from $\nabla c$

$c_l < r + 1$ : $r$ was chosen to be greater than any $c_l$.

$c_l < c_i$ : The $\nabla c$ is the same for both sets of values.

$c_l < d'_k$ : This will also be true of $c_l < d_k$:

* If $d_k < r$, then $d_k = d'_k$, so $c_l < d_k$.
* If $d_k > r$, $d'_k = d_k + 1$. Since $c_l < r$ and $d_k \geq r$, $c_l < d_k$.
* $d_k = r$ will never occur in the scheme outlined above.

Either way, $c_l < d_k$.

$c_l < e'_h$ : This will also be true of $c_l < e_h$:

* If $e'_h \geq r$, then all $c_l$ will be less than $e_h$ because $c_l < r$.
* If $e'_h < r$, then $e'_h < r + 1$, so $e'_h = e_h$, so $c_l < e_h$.

- Sentences starting with an element from $\nabla d$

$d'_k < r + 1$ : If $d'_k < r + 1$, $d_k = d'_k$ and $d_k + 1 \neq r + 1$, so $d_k < r$.

$d'_k < c_l$ : Every $c_l$ is less than $r$, so if $d'_k < c_l$, $d_k = d'_k$ and thus $d_k < c_l$.

$d'_k < d'_i$ : This will also be true of $d_k < d_i$:

* If $d'_k < d'_i < r$, $d'_k = d_k$ and $d'_i = d_i$, so $d_k < d_i$.
* If $r < d'_k < d'_i$, $d'_k = d_k + 1$ and $d'_i = d_i + 1$, so $d_k < d_i$.
* If $d'_k < r < d'_i$, there are an infinite number of elements between $d'_k$ and $d'_i$, since $r$ is a limit. So, $d'_k + 1 < d'_i$, since $d'_i = d_i + 1$ and $d'_k = d_k$, $d_k + 1 < d_i + 1$, so $d_k < d_i$. 

15
Since \( r \) is a limit, \( d' \) will never be equal to \( r \).

\[ d' < e'_h : \text{This will also be true of } d_k < e_h: \]

* If \( d' < e'_h < r + 1, d_k = d' \) and \( e'_h = e_h \), so \( d_k < e_h \).

* If \( r + 1 < d' < e'_h, d' = d_k + 1 \) and \( e_h \) is either such that \( e'_h = e_h + 1 \)
or chosen to be greater than \( d'_k \).

- Sentences starting with an element from \( e' \)

\[ e'_h < r + 1 : \text{If } e'_h = r, e_h \text{ will be chosen to be less than } r. \text{ If } e'_h < r, e'_h = e_h, \text{ so } e_h < r. \]

\[ e'_h < c_l : \text{Each } e_h \leq e'_h, \text{ so } e_h < c_l. \]

\[ e'_h < d'_k : \text{This will also be true of } e_h < d_k: \]

* If \( e'_h < d'_k < r \), then \( e_h = e'_h \) and \( d_k = d'_k \), so \( e_h < d_k \).

* If \( r \leq e'_h < d'_k \), then \( e_h + 1 \leq e'_h \) and \( d_k + 1 = d'_k \), so \( e_h + 1 < d_k + 1, \)
  so \( e_h < d_k \).

* If \( e'_h < r < d'_k \), then \( e_h = e'_h < r \) and \( r < d_k + 1 = d'_k \), so \( r \leq d_k, \)
  so \( e_h < d_k \).

\[ e'_h < e'_i : \text{This will also be true of } e_h < e_i: \]

* If \( e'_h < e'_i < r \), then \( e'_h = e_h \) and \( e'_i = e_i \), so \( e_h < e_i \).

* If \( e'_h < e'_i = r \), then \( e_i \) is chosen to be greater than any \( e'_h < r \), and \( e_h = e'_h \), so \( e_h < e_i \).

* If \( r \leq e'_h < e'_i, e_h < e'_h. \) If \( e'_i \) is a limit, \( e_i \) is greater than any \( e'_h < e'_i \), so \( e_h < e'_i \). Otherwise, \( e_i + 1 = e'_i, \text{ so } e'_h \leq e_i, \text{ so } e_h < e_i. \)

* If \( e'_h < r \leq e'_i, e_h = e'_h. \) If \( r = e'_i, \text{ then } e_i \) was chosen to be greater than \( e'_h \). Otherwise, \( r < e_i + 1 = e'_i, \text{ so } e_h < r \leq e_i, \text{ so } e_h < e_i. \)
Consider sentences of the form \( x = y \). Note that, if both sides of the equation are from the same tuple, the sentence is vacuously true, since both sides will have been chosen by the same scheme.

- Sentences starting with \( r + 1 \)

  \( r + 1 = c_l \) : These sentences will always be false, since \( r \) was chosen to be greater than any \( c \).

  \( r + 1 = d'_k \) : Assume \( d_k = d'_k \). This means \( d_k < r \). But that means \( d'_k < r \), so \( d'_k \neq r + 1 \). Thus, \( d_k + 1 = d'_k = r + 1 \), so \( d_k = r \).

  \( r + 1 = e'_h \) : This means \( e'_h \) is not a limit, so \( e_h + 1 = e'_h = r + 1 \), so \( e_h = r \).

- Sentences starting with elements from \( \overrightarrow{c} \)

  \( c_l = d'_k \) : Each \( c_l < r \), so \( d_k = d'_k = c_l \).

  \( c_l = e'_h \) : Again, each \( c_l < r \), so \( e_h = e'_h = c_l \).

- Sentences starting with elements from \( \overrightarrow{d} \)

  \( d'_k = e'_h \) : This will also be true of \( d_k = e_h \). Note that \( d'_k \) is never equal to \( r \), and is never a limit if greater than \( r \):

    * If \( d'_k = e'_h < r \), \( d_k = d'_k \) and \( e_h = e'_h \), so \( d_k = e_h \).

    * If \( r < d'_k = e'_h \), \( d_k + 1 = d'_k \) and \( e_h + 1 = e'_h \), so \( d_k = e_h \).

Therefore, since every atomic sentence true of \( r+1 \), \( \overrightarrow{c} \), \( \overrightarrow{d} \), \( \overrightarrow{e} \) is true of \( r \), \( \overrightarrow{c} \), \( \overrightarrow{d} \), \( \overrightarrow{e} \), every conjunction of disjunctions will be true, which means every \( \Sigma^0_2 \) formula that defines \( r \) can also define \( r + 1 \), and is therefore incapable of picking out limit points.
2.1.2 The General Case

More generally, for $J \in \omega$, define $R_J$ with the formula

$$\rho_J(x) = \left( \exists y \left( y < x \land \rho_{n-1}(y) \right) \land \left( \forall y \left( \exists z \left( y < x \land \rho_{n-1}(y) \Rightarrow y < z \land z < x \land \rho_{J-1}(z) \right) \right) \right)$$

So, $R_1$ picks out limits, $R_2$ picks out limits of limits, $R_3$ picks out limits of limits of limits, et cetera. For each finite $J$ and each $N > J$, $R_J$ is computably enumerable in automatic copies of $\langle \omega^N, < \rangle$, as given by Theorem 1.2.9. However, $R_J$ is defined with a $\Pi^0_{2J}$ formula, so can be substantially more complicated in general computable copies of $\langle \omega^N, < \rangle$.

Additional lemmas are needed to prove that $R_J$ is not intrinsically $\Sigma^0_{2J}$.

2.1.2.1 The Back and Forth Relations for General Linear Orderings

**Lemma 2.1.2.** Suppose $\mathcal{A}$ and $\mathcal{B}$ are linear orderings. Let $\vec{a} = (a_0, a_1, \ldots, a_n)$, $\vec{b} = (b_0, b_1, \ldots, b_n)$ be increasing tuples from $\mathcal{A}$ and $\mathcal{B}$ respectively. For $0 \leq i \leq n + 1$, let $\mathcal{A}_i$ and $\mathcal{B}_i$ such that

$$\mathcal{A} = \mathcal{A}_0 + \{a_0\} + \mathcal{A}_1 + \{a_1\} + \ldots + \mathcal{A}_n + \{a_n\} + \mathcal{A}_{n+1}$$

$$\mathcal{B} = \mathcal{B}_0 + \{b_0\} + \mathcal{B}_1 + \{b_1\} + \ldots + \mathcal{B}_n + \{b_n\} + \mathcal{B}_{n+1}$$

Then $\langle \mathcal{A}, \vec{a} \rangle \leq_\beta \langle \mathcal{B}, \vec{b} \rangle$ iff, for $0 \leq i \leq n + 1$, $\mathcal{A} \leq_\beta \mathcal{B}$. [2]

2.1.2.2 The Back and Forth Relations for Ordinals

A common problem involving computable ordinals $\alpha$, $\beta$, and $\gamma$ is determining whether or not $\alpha \leq_\gamma \beta$. There is another lemma from [2] useful in this case,
although it takes a bit of preparation to state. The lemma has to break into three cases, based on whether the finite part of $\gamma$ is odd, even, or zero. Find $\delta$ a limit ordinal and $\mu$ a finite number such that $\gamma = \delta + 2\mu + 1$, $\gamma = \delta + 2\mu + 2$, or $\gamma = \delta$.

For any $\xi$, $\alpha$ and $\beta$ can be written in the following form:

$$\alpha = \omega^\xi \cdot \alpha_\xi + \rho_\xi$$

$$\beta = \omega^\xi \cdot \beta_\xi + \sigma_\xi$$

where $\rho_\xi, \sigma_\xi < \omega^\xi$. Note that, if $\alpha < \omega^\xi$, then $\alpha_\xi = 0$ and $\rho_\xi = \alpha$; similarly for $\beta$. This notation breaks up $\alpha$ and $\beta$ into parts that are above and below a given limit ordinal.

The other notation the lemma needs for $\alpha$ and $\beta$ is Cantor normal form. In this form,

$$\alpha = \sum_{\xi=0}^{A} \omega^\xi \cdot m_\xi \text{ and } \beta = \sum_{\xi=0}^{B} \omega^\xi \cdot n_\xi$$

for some $m_0, m_1, \ldots \in \omega$ and $n_0, n_1, \ldots \in \omega$. In general, $A$ and $B$ can be any ordinal, which is why the $m$’s and $n$’s can be restricted to finite values.

The lemma only involves two cases: the case where $\xi = \mu$ and the case where $\xi = \mu + 1$.

**Lemma 2.1.3.** Using the notation described above,

1. $\alpha \leq_{\delta+2\mu+1} \beta$ iff one of the following holds:

   (a) $\alpha = \beta$

   (b) $\rho_{\delta+\mu} = \sigma_{\delta+\mu}$, $\alpha_{\delta+\mu+1} \geq 1$ and $\beta_{\delta+\mu+1} \geq 1$

   (c) $\rho_{\delta+\mu} = \sigma_{\delta+\mu}$, $\alpha_{\delta+\mu+1} = \beta_{\delta+\mu+1} = 0$, and $m_{\delta+\mu} \geq n_{\delta+\mu} > 0$
2. \( \alpha \leq_{\delta+2\mu+2} \beta \) iff one of the following holds:

(a) \( \alpha = \beta \)

(b) \( \rho_{\delta+\mu} = \sigma_{\delta+\mu}, \alpha_{\delta+\mu+1} \geq 1, \beta_{\delta+\mu+1} \geq 1 \) and \( m_{\delta+\mu} \geq n_{\delta+\mu} \)

3. \( \alpha \leq_{\delta} \beta \) iff one of the following holds:

(a) \( \alpha = \beta \)

(b) \( \rho_{\delta} = \sigma_{\delta}, \alpha_{\delta} \geq 1 \) and \( \beta_{\delta} \geq 1 \)

Note that the only \( \alpha_{\xi} \) and \( \beta_{\xi} \) considered are \( \alpha_{\delta+\mu+1} \) and \( \beta_{\delta+\mu+1} \), whereas the only \( \sigma_{\xi}, \rho_{\xi}, m_{\xi} \) and \( n_{\xi} \) considered are \( \sigma_{\delta+\mu}, \rho_{\delta+\mu}, m_{\delta+\mu} \) and \( n_{\delta+\mu} \).

2.1.2.3 The \( J \)-Limits in Ordinals

**Theorem 2.1.4.** Let \( N \in \omega \), and \( J < N \). Then, in the structure \( \langle \omega^N, < \rangle \), \( R_J \) is not relatively intrinsically \( \Sigma^0_{2J} \).

**Proof.** Assume there is some \( \Sigma^0_{2J} \) formula \( \Phi(x, \overrightarrow{c}) \) that defines \( \langle \omega^N, < \rangle \). Without loss of generality, assume that the \( c \)'s are indexed in increasing order. Let \( k \) be the number of \( c \)'s. Let \( r \) be a \( J \)-limit greater than any of the \( c \)'s that is not also a \( J+1 \) limit, and let \( r' = r + \omega^{J-1} \). Note that \( r' \) is not a \( J \)-limit. By 1.2.1, if \( \langle \omega^N, < \rangle, \overrightarrow{c}, r' \rangle \leq_{2J} \langle \omega^N, < \rangle, \overrightarrow{c}, r \rangle \), then \( \langle \omega^N, < \rangle \models \Phi(r', \overrightarrow{c}) \).

Applying 2.1.2, note that all of the intervals \([0, c_i), [c_i, c_{i+1})\) are the same in \( \langle \omega^N, < \rangle, \overrightarrow{c}, r' \rangle \) as they are in \( \langle \omega^N, < \rangle, \overrightarrow{c}, r \rangle \). Likewise, the interval after \( r \) is equal to the interval after \( r' \) is equal to \( \omega^N \). So, the only interval that differs between the two structures is \([c_k, r) \) and \([c_k, r') \). If \([c_k, r') \leq_{2J} [c_k, r) \), then \( \langle \omega^N, < \rangle \models \Phi(r', \overrightarrow{c}) \).

20
Let \( \alpha = [c_k, r'] \) and \( \beta = [c_k, r] \). Since \( r \) is a \( J \)-limit, \( \beta = \omega^J \cdot u \) for some natural number \( u \geq 1 \). Since \( r' = r + \omega^{J-1} \), \( \alpha = \omega^J \cdot u + \omega^{J-1} \). Let \( \mu = J - 1 \), so \( J = \mu + 1 \) and \( 2J = 2\mu + 2 \). Then, using the notation in 2.1.3 (with \( \delta = 0 \)),

\[
\alpha = \omega^{\mu+1} \cdot \alpha_{\mu+1} + \rho_{\mu+1} = \omega^\mu \cdot \alpha_\mu + \rho_\mu
\]

In this case, \( \alpha_{\mu+1} = u \), \( \rho_{\mu+1} = \omega^\mu \), \( \alpha_\mu = (\omega + 1) \), \( \rho_\mu = 0 \), and \( m_\mu = 1 \).

\[
\beta = \omega^{\mu+1} \cdot \beta_{\mu+1} + \sigma_{\mu+1} = \omega^\mu \cdot \beta_\mu + \sigma_\mu
\]

In this case, \( \beta_{\mu+1} = u \), \( \sigma_{\mu+1} = 0 \), \( \beta_\mu = \omega \cdot u \), \( \sigma_\mu = 0 \), and \( n_\mu = 0 \).

For \( \alpha \leq 2_{\mu+2} \beta \) to hold, we need four things:

- \( \rho_\mu = \sigma_\mu \) This holds because \( \rho_\mu = \sigma_\mu = 0 \).
- \( \alpha_{\mu+1} \geq 1 \) This holds because \( \alpha_{\mu+1} = u \geq 1 \).
- \( \beta_{\mu+1} \geq 1 \) This holds because \( \beta_{\mu+1} = u \geq 1 \).
- \( m_\mu \geq n_\mu \) This holds because \( m_\mu = 1 \) and \( n_\mu = 0 \).

Thus, \( \alpha \leq 2_{\mu+2} \beta \), which is the same as saying \([c_k, r'] \leq_{2J} [c_k, r] \). This means the \( \Sigma^0_{2J} \) sentences true of \((\overline{c'}, r)\) are true of \((\overline{c'}, r')\); in particular, \( \langle \omega^N, < \rangle \models \Phi(r', \overline{c'}) \).

Since \( r' \) is not a \( J \)-limit, no \( \Sigma^0_{2J} \) sentence can define the \( J \)-limits.

2.2 Categoricity of \( \omega^n \)

In [\[\Pi\]], it was shown that for finite \( n \geq 1 \), \( \omega^n \) is strictly \( \Delta^0_{2n} \)-stable. For any two automatic copies, there is guaranteed to be a computable isomorphism between them.
Lemma 2.2.1. For finite \( n \), \( \omega^n \) is rigid.

Proof. Let \( \alpha = \omega^A \cdot m_A + \omega^{A-1} \cdot m_{A-1} + \ldots + \omega \cdot m_1 + m_0 \) be an element of \( \omega^n \).

Define the formula \( \psi_\alpha \) as follows.

There exist \( m_A \) unique \( y_{A_i} \), there exist \( m_{A-1} \) unique \( m_{(A-1)_i} \), \ldots, there exist \( m_1 \) unique \( y_{1_i} \), and there exist \( m_0 \) unique \( y_{0_i} \) such that:

- For each \( j < A \) and \( i < m_j \), we have \( \rho_j(y_{j_i}) \)
- For each \( h < A, k < h, \) and each \( i < m_h \), we have each \( y_{h_i} < y_{k_i} \).

Then \( \alpha \) is uniquely defined by the formula \( \phi(x) = \psi_\alpha(x) \land \neg \psi_{\alpha+1}(x) \). Since \( n \) is finite, \( A \) is finite, so \( \phi(x) \) is a finitary formula. So, the set of \( \phi \)'s makes a defining family for \( \omega^n \); by Theorem 1.2.4, this means \( \omega^n \) is rigid. \( \square \)

Note that the set of formulas \( \phi \) in the above proof is computably enumerable.

Lemma 2.2.2. Let \( \mathcal{A} \) be an automatic rigid structure with defining family \( \Phi \). If \( \Phi \) is computably enumerable, then for any automatic isomorphic copy \( \mathcal{B} \), there is a computable isomorphism \( f \) from \( \mathcal{A} \) to \( \mathcal{B} \).

Proof. For any \( a \in \mathcal{A} \), to compute \( f(a) \)

1. Enumerate formulas of \( \Phi \) until you find a formula \( \phi \) such that \( \mathcal{A} \models \phi(a) \).
2. Enumerate elements of \( \mathcal{B} \) until you find \( b \) such that \( \mathcal{B} \models \phi(b) \). Define \( f(a) = b \).

Since \( \mathcal{A} \) and \( \mathcal{B} \) are assumed to be automatic, checking whether an element satisfies a formula is computable. Since no formula of \( \Phi \) is satisfied by two distinct elements in the respective structures, you don’t even need to go back-and-forth; each \( \phi \) will pick out exactly one element. \( \square \)
**Theorem 2.2.3.** For finite $n$, if $A$ and $B$ are automatic copies of $\omega^n$, there exists a computable isomorphism from $A$ to $B$.

*Proof.* Follows immediately from Lemmas 2.2.1 and 2.2.2. □
Douglas Cenzer started the investigation into automatic equivalence structures. He created an automatic structure with infinitely many equivalence classes of size one and infinitely many classes of size two. This structure is $\Delta^0_2$ categorical and not computably categorical, but between any two automatic copies there is a computable isomorphism. He made some conjectures about the existence of other equivalence structures with similar properties.

3.1 General Facts about Automatic Equivalence Structures

An equivalence structure is any structure $\mathcal{A} = \langle A, R \rangle$ where $R$ is an equivalence relation. Usually, when writing a formula using an equivalence relation, the symbol for the relation is written between the two variables being compared (as in $a R b$) instead of using the normal parentheses-based notation ($R(a, b)$). The symbol $\sim$ will be used for equivalence relations defined in the structure in this dissertation.

3.1.1 Definitions and Notation

**Definition 16.** Let $[x]$ be the $\sim$-equivalence class of $x$.

**Definition 17.** Let $\mathcal{A}$ be an equivalence structure. $\mathcal{A}$ is said to have bounded character iff there is some finite $k$ such that all finite equivalence classes in $\mathcal{A}$
have size at most $k$.

3.1.2 Theorems

The following theorems about equivalence structures are due to [3].

**Lemma 3.1.1** (Calvert, Cenzer, Harizanov, Morozov). Let $\mathcal{A}$ be a computable equivalence structure. If $\mathcal{A}$ is computably categorical, then $\mathcal{A}$ is relatively computably categorical.

Recall that relative computable categoricity implies computable categoricity.

**Theorem 3.1.2.** Let $\mathcal{A}$ be a computable equivalence structure. Then $\mathcal{A}$ is relatively computably categorical if and only if one of the following holds.

1. $\mathcal{A}$ has finitely many finite equivalence classes.
2. $\mathcal{A}$ has finitely many infinite classes, bounded character, and at most one finite $k$ such that there are infinitely many classes of size $k$.

**Theorem 3.1.3.** Let $\mathcal{A}$ be a computable equivalence structure. Then $\mathcal{A}$ is relatively $\Delta^0_2$-categorical if and only if one of the following holds.

1. $\mathcal{A}$ has finitely many infinite equivalence classes.
2. $\mathcal{A}$ has bounded character.

**Theorem 3.1.4.** Let $\mathcal{A}$ be a computable equivalence structure. Then $\mathcal{A}$ is relatively $\Delta^0_3$-categorical.
3.2 Constructing an Automatic Equivalence Structure

3.2.1 Describing the Desired Equivalence Structure

The specific structure we want will have infinitely many finite equivalence classes. There will only be one of each size, although it does not matter if some sizes are not represented. It will also have infinitely many equivalence classes of infinite size. In honor of the infinitely many infinite classes, let $\mathcal{A}_\infty$ denote this structure.

In addition, the structure should be automatic. So, its universe $\mathcal{A}_\infty$ will consist of strings over some alphabet $\Sigma$. Choosing the alphabet $\Sigma = \{0, 1\}$ will be sufficient. The universe itself is the set of nonempty strings over $\Sigma$. Constructing an automaton that recognizes $\mathcal{A}_\infty$ is left as an exercise for the reader.

The goal in creating $\mathcal{A}_\infty$ is to end up with a new example of an automatic equivalence structure that is as complicated as any equivalence structure can be. It will also be a simpler version of the structure examined in Chapter 4.

3.2.1.1 Assigning Strings to Equivalence Classes

Let $\sigma$ be any string of 0’s and 1’s of length at least 1. The first character of $\sigma$ will determine what type of equivalence class $\sigma$ ends up in. If it is 0, $\sigma$ will be in a finite class; if it is 1, $\sigma$ will be in an infinite class.

If $\sigma$ is in a finite class, it will be equivalent to any other string of the same length. Thus, the finite equivalence classes of $\mathcal{A}_\infty$ will have size $2^n$ for some $n \in \omega$, and every power of two will have a class of that size.

If $\sigma$ is in an infinite class, consider the length of the string of 0’s starting from the second character all the way to either the next 1 in the string or the end of the string. $\sigma$ will be equivalent to any other string for which that length is the
same. So, for example,

\[ 100001001 \sim 10000101101110111011111 \sim 10000 \]

Of course, if the second character of \( \sigma \) is a 1, that corresponds to a string of 0’s of length 0.

These criteria produce an equivalence relation that is automatic on pairs of elements, as illustrated by figure 3.1.

3.2.2 Warming Up with a Single Infinite Class

Let \( A_1 \) be a substructure of \( A_\infty \) that has all the finite equivalence classes of \( A_\infty \), but just one of the infinite equivalence classes. \( A_1 \), the universe of \( A_1 \), is just the union of the set of strings over \( \Sigma \) that start with 0 with the set of strings over

![Figure 3.1](image-url)

Figure 3.1. The automaton that recognizes the equivalence relation of \( A_\infty \). The symbol on the bottom-right loop is shorthand for the list of all possible symbols; once the machine enters the state with the loop with that label, it is guaranteed to accept the string it reads.
Σ that start with 11.

3.2.2.1 $\Delta^0_2$-Categoricity

**Theorem 3.2.1.** $A_1$ is relatively $\Delta^0_2$-categorical.

**Proof.** For any element $a$ of $A_1$, the only information about $a$ that matters is the equivalence class it is in; specifically, the size of the equivalence class that contains $a$.

For any tuple $\vec{x}$, let $A(\vec{x})$ be the set of formulas

$$
\left\{ \bigwedge_{i<j \leq |\vec{x}|} R_{i,j}(x_i, x_j) \bigg| R_{i,j} \in \{=, \neq, \sim\} \right\}
$$

The set $A$ basically consists of all formulas that talk about how individual elements of a tuple are related to each other. Each formula in $A$ is $\Sigma^0_0$.

For any $n \in \omega$, let $s_n(x)$ be the formula

$$
\left( \exists y_1, \ldots, y_n \right) \left( \bigwedge_{i<j \leq n} y_i \sim x \land y_i \neq y_j \land \left( \forall z \left( z \sim x \rightarrow \bigvee_{k \leq n} y_k = z \right) \right) \right)
$$

![Figure 3.2. The machine that recognizes the universe of $A_1$.](image-url)
The formula $s_n(x)$ is therefore a $\Sigma^0_2$ formula saying that $x$ is in an equivalence class with exactly $n$ elements. Let $s_\omega(x)$ be the formula

$$\bigwedge_{n \in \omega} \left( \exists y_1, \ldots, y_n \left( \bigwedge_{i < j \leq n} y_i \sim x \land y_i \neq y_j \right) \right)$$

Thus, the formula $s_\omega(x)$ is a $\Pi^0_2$ formula saying that $x$ is in an equivalence class with arbitrarily many elements. For any tuple $\bar{x}$, let $\mathcal{S}(\bar{x})$ be the set of formulas

$$\left\{ \left. \bigwedge_{i \leq \bar{x}} s_{n_i}(x_i) \right| n_i \in (\omega + 1) \right\}$$

The set $\mathcal{S}(\bar{x})$ is the set of all possible statements about the sizes of the equivalence classes of the elements of $\bar{x}$.

Let $\Phi$ be the set of formulas

$$\{ \alpha(\bar{x}) \land s(\bar{x}) | \alpha \in \mathcal{A}(\bar{x}), s \in \mathcal{S}(\bar{x}) \}$$

where $\bar{x}$ ranges over tuples of variables of any length. Let $\bar{a}$ be any tuple from $\mathcal{A}_1$. There will be some $\alpha \in \mathcal{A}(\bar{a})$ such that $\mathcal{A}_1 \models \alpha(\bar{a})$, since $\mathcal{A}$ just states the relations between elements. In addition, $\mathcal{A}_1 \models \bigwedge_{i \not\in \{\bar{x}\}} s_{|a_i|}(a_i)$, which is a sentence made from a formula in $\mathcal{S}(\bar{a})$. Furthermore, since $\mathcal{A}_1$ only has one equivalence class of each size, if $\bar{b}$ is a tuple of distinct elements such that $\mathcal{A}_1 \models \alpha(\bar{b}) \land \bigwedge_{i \not\in \{\bar{x}\}} s_{|a_i|}(b_i)$, then for each $i$, $b_i$ is in the same equivalence class as $a_i$, and the atomic information about $\bar{a}$ is true of $\bar{b}$. Thus, $(\mathcal{A}_1, \bar{a}) \cong (\mathcal{A}_1, \bar{b})$.

Therefore, $\Phi$ is a Scott family for $\mathcal{A}_1$. Since each element of $\Phi$ is a $\Sigma^0_2$ formula, $\Phi$ is a formally $\Sigma^0_2$ Scott family for $\mathcal{A}_1$. Finally, by Lemma 1.2.2, $\mathcal{A}_1$ is relatively $\Delta^0_2$-categorical. \qed
3.2.2.2 Not Computably Categorical

**Theorem 3.2.2.** $\mathcal{A}_1$ is not computably categorical.

**Proof.** Theorem 3.1.2 partitions the relatively computably categorical equivalence structures into two types. $\mathcal{A}_1$ is not of the first type, since $\mathcal{A}_1$ has infinitely many finite equivalence classes. $\mathcal{A}_1$ is not of the second type either; $\mathcal{A}_1$ has finite equivalence classes of size $2^n$ for all $n \in \omega$, so $\mathcal{A}_1$ does not have bounded character. Thus, $\mathcal{A}_1$ is not relatively computably categorical. By Lemma 3.1.1, since $\mathcal{A}_1$ is not relatively computably categorical, it can not even be computably categorical. \qed

3.2.3 Completing the Structure with Infinitely Many Classes

Consider again the complete structure $\mathcal{A}_\infty$. $\mathcal{A}_\infty$ is computable, so it is automatically $\Delta^0_3$-categorical by Theorem 3.1.4. Since $\mathcal{A}_\infty$ has infinitely many infinite equivalence classes and unbounded character, $\mathcal{A}_\infty$ cannot be $\Delta^0_2$-categorical. So, as far as the arithmetical hierarchy can tell, automatic equivalence structures can be as complicated as any equivalence structure can be.
CHAPTER 4

AUTOMATIC NESTED EQUIVALENCE STRUCTURES

A nested equivalence structure is a structure $\mathcal{A} = \langle A, \sim_1, \sim_2, \ldots \rangle$ where each of the $\sim$’s is an equivalence relation, and for $m < n$, $x \sim_m y \rightarrow x \sim_n y$. The indices start at 1 so that it is convenient to think of $\sim_0$ as genuine equality. Nested equivalence structures are obviously similar to equivalence structures, but they are not restricted to being only $\Delta_3^0$-categorical.

Definition 18. For any $a$ in the universe of a nested equivalence structure, let $[a]_n$ be the size of the $\sim_n$-equivalence class of $a$.

4.1 The Desired Nested Equivalence Structure

The goal of this section is to create an automatic nested equivalence structure that is not $\Delta_3^0$-categorical. To do this, let $\mathcal{A}_2$ be a nested equivalence structure with two relations, satisfying the following properties.

1. Each $\sim_2$-class contains at least one infinite $\sim_1$-class.

2. Each $\sim_2$-class contains finite $\sim_1$-classes of arbitrarily large size.

3. For each $j$ such that a $\sim_2$-class contains a finite $\sim_1$-class of size $j$, every $\sim_2$-class contains a finite $\sim_1$-class of size $j$. 
4. For each $\sim_2$-class and each $j$, if the $\sim_2$-class contains a finite $\sim_1$-class of size $j$, then it contains only one finite $\sim_1$-class of size $j$.

4.1.1 Assigning Strings to Equivalence Classes

The universe of $A_2$ is more complicated than the universes of $A_1$ or $A_\infty$. As in those cases, the first character of a string will determine whether it is in a finite $\sim_1$-class or an infinite $\sim_1$-class. For the strings that start with 0, the rest of the string must be a positive number of 0’s, followed by a 1, followed by any number (including zero) of 0’s, followed by any number (including zero) of 1’s. For the strings that start with 1, there must be at least one more character in the string, but that is the only restriction. The reasons for the strange universe will become clear when explaining the equivalence relations. $A_2$ is automatic; it is recognized by the machine in Figure 4.1.

Figure 4.1. The automaton that recognizes the universe of $A_2$
4.1.1.1 Finding the Equivalence Class of a String

Let $\sigma$ be a string in $A_2$. If the first character is a 0, $\sigma$ is in a finite $\sim_1$-class; if the first character is a 1, $\sigma$ is in an infinite $\sim_1$-class.

If the first character of $\sigma$ is a 0, there will be a positive number $h$ of 0’s in a row before the first 1 in the string. $h$ will also be the number of infinite $\sim_1$-classes in the same $\sim_2$-class as $\sigma$. The rest of $\sigma$ will be a string of the form $0^j1^k$, where $j$ or $k$ can be zero. $\sigma$ will be in the finite $\sim_1$-class with $j+k+1$ other elements. For example,

$$00000010011 \sim_1 00000011111 \sim_1 00000010000$$

A string like $0000001$ will be in a $\sim_1$-class by itself.

If the first character of $\sigma$ is a 1, consider the next substring of the form $0^m1^n$ (or $0^m1^n$ if the string terminates immediately afterwards). It is possible for $m$ to be zero. $n$ can only be zero if the string terminates immediately afterwards, and only if $m$ is not also zero. The total $m+n$ will be equal to the number of infinite $\sim_1$-classes in the same $\sim_2$-class as $\sigma$. $\sigma$ will be in the $\sim_1$-equivalence class with every other string that has the same values for $m$ and $n$. So, for example,

$$100011101101001 \sim_1 100011101011011110111110111111 \sim_1 1000111$$

$$110100100010000 \sim_1 1101101001 \sim_1 11$$

4.1.1.2 Finite Automata that Recognize the Equivalence Relations

A machine that decides whether or not two strings are $\sim_1$-equivalent will first have to make sure that the first two characters of the strings are equal. If they are
both 0, the machine will then have to ensure that the two strings have the same values for $h$, then that they both have the same values for $j + k$. If all conditions are met, the two strings will be $\sim_1$-equivalent; otherwise, they are not. If the first character of both strings in 1, the machine will need to make sure that the two strings have the same values for $m$ and $n$. If both pairs of values are equal, the two strings are $\sim_1$-equivalent, otherwise not. A finite automaton that does this is in figure 4.2.

A machine that decides whether or not two strings are $\sim_2$-equivalent will have to check different cases based on the first characters of the strings. If they are both 0, the machine will have to determine that the two strings have the same values for $h$. If they are both 1, the machine will have to determine if the two strings have the same values for $m + n$. If one is 0 and the other is 1, the machine will have to determine if the string starting with 0’s value for $h$ is the same as

![Figure 4.2. A finite automaton that recognizes $\sim_1$.](image-url)
the string starting with 1’s value for \( m + n \). If the machine determines that the answer is yes, the two strings are \( \sim_2 \)-equivalent; otherwise, they are not. Such a machine is shown in figure 4.3.

4.2 Proving Categoricity of the Constructed Structure

4.2.1 The Back and Forth Relations

As it happens, in \( A_2 \), the back-and-forth relations are equivalent to conditions that are computable. Defining the back-and-forth relations in \( A_2 \) will be easier after borrowing notation from [8].

Figure 4.3. A finite automaton that recognizes \( \sim_2 \).
4.2.1.1 Useful Notation

**Definition 19.** Given tuples $\bar{x}$ and $\bar{y}$ from a nested equivalence structure $\mathcal{B}$, $\bar{x} \equiv_0 \bar{y}$ iff the following three conditions hold.

1. $\{\bar{x}\} = \{\bar{y}\}$

2. For each $i, j \leq \{x\}$, $\mathcal{B} \models x_i = x_j$ iff $\mathcal{B} \models y_i = y_j$.

3. For each $n$ and each $i, j \leq \{x\}$, $\mathcal{B} \models x_i \sim_n x_j$ iff $\mathcal{B} \models y_i \sim_n y_j$.

Essentially, the expression $\bar{x} \equiv_0 \bar{y}$ states that the atomic information true of $\bar{x}$ is also true of $\bar{y}$.

**Definition 20.** Let $[x]_1$ be the number of infinite $\sim_1$-classes in the same $\sim_2$-class as $x$.

4.2.1.2 $\leq_1$

**Claim 1.** For any tuples $\bar{x}$ and $\bar{y}$ of the same length of elements in $\mathcal{A}_2$, $\bar{x} \leq_1 \bar{y}$ iff both of the following conditions hold:

1. $\bar{x} \equiv_0 \bar{y}$

2. $|[y_i]_1| \leq |[x_i]_1|$ for each $y_i \in \bar{y}$ and each $x_i \in \bar{x}$.

**Proof.** ($\Rightarrow$) This direction shall be proved using the contrapositive of the statement. Assume $\neg \bar{x} \equiv_0 \bar{y}$. Since the tuples are stated to be the same length, there are some $i$ and $j$ such that $\mathcal{A}_2 \models x_i = x_j$ but $\mathcal{A}_2 \models y_i \neq y_j$ or vice-versa, or $\mathcal{A}_2 \models x_i \sim_n x_j$ but $\neg \mathcal{A}_2 \models y_i \sim_n y_j$ or vice-versa. Either way, the offending formula will be true of $\bar{y}$ but not $\bar{x}$, so $\bar{x} \not\leq_1 \bar{y}$. 

36
Assume there is some $i$ such that $| [y_i]_1 | > | [x_i]_1 |$. Let $\mathcal{D}_1(w)$ be the formula

$$\exists z_1, \ldots, z_n \left( \bigwedge_{i<j \leq n} (w \sim_1 z_i \land z_i \neq z_j) \right)$$

The formula $\mathcal{D}_1(w)$ then says there are at least $n$ elements that are $\sim_1$-equivalent to $w$ but not pairwise equal to each other. Thus, $\mathcal{A}_2 \models \mathcal{D}_1([y_i]_1)(y_i)$ but $\mathcal{A}_2 \not\models \mathcal{D}_1([x_i]_1)(x_i)$, since $| [x_i]_1 | < | [y_i]_1 |$, so $\overrightarrow{x} \notin \overrightarrow{y}$.

Therefore, if $\overrightarrow{x} \leq \overrightarrow{y}$, then $\overrightarrow{x} \equiv_0 \overrightarrow{y}$ and $| [y_i]_1 | \leq | [x_i]_1 |$ for each $y_i \in \overrightarrow{y}$ and each $x_i \in \overrightarrow{x}$.

$(\Leftarrow)$ Let $\Phi(\overrightarrow{x}, \overrightarrow{c})$ be a $\Sigma^0_1$ formula true of $\overrightarrow{y}$. $\Phi$ is of the form $\bigvee_i (\exists u_i)(\phi_i(\overrightarrow{x}, \overrightarrow{c}, u_i))$, where each $\phi_i$ is a conjunction of literals. Let $I$ be an index for a disjunct true of $\overrightarrow{y}$ and $\overrightarrow{a}$ a witness for the existential quantifier. Then $\overrightarrow{x} \leq_1 \overrightarrow{y}$ iff there exists $\overrightarrow{c}$ and $\overrightarrow{a}'$ such that $\phi_I(\overrightarrow{x}, \overrightarrow{c}, \overrightarrow{a}')$.

1. For conjuncts of the form $y_i \sim_1 y_j$, $y_i \sim_2 y_j$, $y_i = y_j$, or their negations, this will also be true of $x_i$ and $x_j$, since $\overrightarrow{y} \equiv_0 \overrightarrow{x}$.

2. For conjuncts of the form $y_i \sim_1 a_j$ or $y_i = a_j$, there will be enough elements from $[x_i]_1$ to pick one for $a_j'$ that is not already mentioned, since $| [y_i]_1 | \leq | [x_i]_1 |$.

3. For conjuncts of the form $y_i \sim_1 c_j$ or $y_i = c_j$, again, there will be enough elements from $[x_i]_1$ to pick one for $c_j'$ that is not already mentioned, since $| [y_i]_1 | \leq | [x_i]_1 |$.

4. For conjuncts of the form $y_i \sim_2 a_j$ or $y_i \sim_2 c_j$, save these until the preceding conjuncts have been satisfied; some of these will end up satisfied anyway because $\sim_1$-equivalence implies $\sim_2$-equivalence. For those that
are not satisfied, there will be an infinite number of elements to pick from to determine $a'_j$ or $c'_j$, since $\sim_2$-equivalence classes are infinitely large.

5. For conjuncts of the form $y_i \neq c_j, y_i \sim_1 c_j, y_i \sim_2 c_j, y_i \neq a_j, y_i \sim_1 a_j$, or $y_i \sim_2 a_j$, save these until the preceding conjuncts have been satisfied. Again, some will be satisfied anyway because some of the preceding conjuncts will imply these conjuncts. For those that are not satisfied, $a'_j$ or $c'_j$ can be picked from any $\sim_2$-class that contains no $y$’s to make the conjunct true.

6. For conjuncts of the form $a_i \sim_1 c_j, a_i \sim_2 c_j, a_i = c_j$, or their negations, since these sentences don’t involve $\bar{y}$, save these until after the other conjuncts are satisfied. If there are still $c'_j$’s or $a'_i$’s that haven’t been assigned, just assign them to make these sentences true.

Thus, if both conditions hold, $\bar{x} \leq_1 \bar{y}$.

\[\Box\]

4.2.1.3 $\leq_2$

Claim 2. For any tuples $\bar{x}$ and $\bar{y}$ of the same length of elements in $A_2$, $\bar{x} \leq_2 \bar{y}$ iff all of the following conditions hold:

1. $\bar{x} \equiv_0 \bar{y}$

2. $|[y_i]_1| = |[x_i]_1|$ for each $y_i \in \bar{y}$ and each $x_i \in \bar{x}$.

Proof. ($\Rightarrow$) Since $\bar{x} \leq_2 \bar{y}$ implies $\bar{x} \leq_1 \bar{y}$, Claim 1 above gives us $\bar{x} \equiv_0 \bar{y}$ and $|[y_i]_1| \leq |[x_i]_1|$ for each $y_i \in \bar{y}$ and each $x_i \in \bar{x}$. Since $\bar{x} \leq_2 \bar{y}$ also

38
implies \( \overrightarrow{y} \leq_1 \overrightarrow{x} \), Claim 1 gives us \( |[x_i]| \leq |[y_i]| \) for each \( x_i \in \overrightarrow{x} \) and each \( y_i \in \overrightarrow{y} \).

(\( \Leftarrow \)) Let \( \Phi(\overrightarrow{z}) \) be a \( \Sigma_2^0 \) formula true of \( \overrightarrow{y} \). \( \Phi \) is of the form \( \bigvee \bigvee \big( \exists \overrightarrow{u} \big( \phi_i(\overrightarrow{z}, \overrightarrow{u}) \big) \big) \), where each \( \phi_i \) is a \( \Pi_1^0 \) formula. Let \( I \) be an index for a disjunct true of \( \overrightarrow{y} \) and \( \overrightarrow{a} \) a witness for the existential quantifier. The equivalence of the first and third parts of Proposition 1.2.1 mean that one can show \( \phi_I(\overrightarrow{y}, \overrightarrow{a}) \) implies \( \phi_I(\overrightarrow{x}, \overrightarrow{a}') \) for some \( \overrightarrow{a}' \) by showing that \( \overrightarrow{y}, \overrightarrow{a} \leq_1 \overrightarrow{x}, \overrightarrow{a}' \). This can be obtained from Claim 1.

We know that \( |[y_i]| \leq |[x_i]| \) for each \( y_i \in \overrightarrow{y} \) and each \( x_i \in \overrightarrow{x} \) from the assumption. We need to show that, for any possible \( \overrightarrow{a} \), there exists \( \overrightarrow{a}' \) such that \( \overrightarrow{y}, \overrightarrow{a} \equiv_0 \overrightarrow{x}, \overrightarrow{a}' \) and \( |[a_i']| \leq |[a_i]| \) for each \( a_i \in \overrightarrow{a} \) and each \( a_i' \in \overrightarrow{a}' \).

To assign values for \( \overrightarrow{a}' \),

1. Sort each \( a_i \) by its \( \sim_1 \)-class, so that each \( a_i \) in the same \( \sim_1 \)-class is listed together.

2. Deal with the lists containing \( a_i \) that are in the same \( \sim_1 \)-class as an element of \( \overrightarrow{y} \): Find the \( \sim_1 \)-class that contains the corresponding elements of \( \overrightarrow{x} \). Assign values in \( \overrightarrow{a}' \) corresponding to those on the list respecting equality/inequality of the pairs of elements from \( \overrightarrow{a} \), between \( \overrightarrow{a} \) and \( \overrightarrow{y} \) as applicable. There will be enough elements to choose from, since \( |[y_i]| = |[x_i]| \) for each \( y_i \in \overrightarrow{y} \) and each \( x_i \in \overrightarrow{x} \).

3. Deal with the lists containing \( a_i \) that are not in the same \( \sim_1 \)-class as any element of \( \overrightarrow{y} \), but are in the same \( \sim_2 \)-class as an element of \( \overrightarrow{y} \), or one of the elements of \( \overrightarrow{a} \) handled in previous steps.

   - If \( a_i \) is in a finite \( \sim_1 \)-class, choose the \( \sim_1 \)-class of the same size
from the ∼\_2\text{-}class containing the y or previously handled a. Note that the selected ∼\_1\text{-}class will have no x’s or other a’\text{’}s in it. If it had any x’s, then a\_i would have been dealt with in the previous step. If it had any other a’\text{’}s in it, then a\_i would have appeared on the list of an a that was dealt with previously.

- If a\_i is in an infinite ∼\_1\text{-}class, choose a ∼\_1\text{-}class that has no y’s or a’s in it that is at least as large as the list of a’s in the same ∼\_1\text{-}class as a\_i.

Either way, assign values in \( \overrightarrow{a} \) from the chosen ∼\_1\text{-}class respecting equality/inequality of the pairs of elements from \( \overrightarrow{a} \).

4. Finally, deal with a list containing an a\_i that is not even in the same ∼\_2\text{-}class as any y, or previously-assigned a. Pick a ∼\_2\text{-}class that contains no x’s and has not had any a’\text{’}s assigned to it yet that has at least as many infinite ∼\_1\text{-}classes as the ∼\_2\text{-}class containing a\_i. Such a ∼\_1\text{-}class will exist because there are ∼\_2\text{-}classes with an arbitrary number of infinite ∼\_1\text{-}classes. Assign values in \( \overrightarrow{a} \) from the chosen ∼\_1\text{-}class in the same manner as in the preceding step.

5. Repeat steps 3–5 until all elements of \( \overrightarrow{a} \) are assigned values.

Since all elements of \( \overrightarrow{a} \) are chosen based on their ∼\_1\text{-}classes first, we can guarantee that the truth of sentences of the form a\_i = a\_j, a\_i \neq a\_j, a\_i \sim\_1 a\_j, and a\_i \sim\_1 a\_j implies the truth of their counterparts in terms of a’\text{’}s and x’s. Step 2 guarantees this quality for sentences of the form a\_i = y\_j and a\_i \sim\_1 y\_j. Step 3 guarantees this for sentences of the form a\_i \sim\_2 y\_j and a\_i \sim\_2 a\_j, as well as most of those of the form a\_i \neq y\_j or a\_i \sim\_1 y\_j. Step 4 guarantees those sentences of the form a\_i \sim\_2 y\_j and a\_i \sim\_2 a\_j, as well as the rest of the
sentences of the form \( a_i \neq y_j \) or \( a_i \sim_1 y_j \) not covered by step 3. We assumed \( \overrightarrow{x} \equiv_0 \overrightarrow{y} \), so all of this can be used to conclude \( \overrightarrow{y}, \overrightarrow{a} \equiv_0 \overrightarrow{x}, \overrightarrow{a'} \).

Finally, note that each \( a'_i \) was chosen so that \( |[a'_i]| \leq |[a_i]| \) for each \( a_i \in \overrightarrow{a} \) and each \( a'_i \in \overrightarrow{a'} \). Therefore, the three conditions listed above are sufficient to conclude \( \overrightarrow{x} \leq_2 \overrightarrow{y} \).

\( \square \)

4.2.1.4 \( \leq_3 \)

**Claim 3.** For any tuples \( \overrightarrow{x} \) and \( \overrightarrow{y} \) of the same length of elements in \( A_2 \), \( \overrightarrow{x} \leq_3 \overrightarrow{y} \) iff all of the following conditions hold:

1. \( \overrightarrow{x} \equiv_0 \overrightarrow{y} \)

2. \( |[y_i]| = |[x_i]| \) for each \( y_i \in \overrightarrow{y} \) and each \( x_i \in \overrightarrow{x} \).

3. \( |y_i| \leq |x_i| \) for each \( y_i \in \overrightarrow{y} \) and each \( x_i \in \overrightarrow{x} \).

**Proof.** \( \Rightarrow \) By Claim 2, the first two conditions are automatically satisfied. Assume there is some \( I \) such that \( |y_I| > |x_I| \). Let \( \mathcal{D}_{2n}(w) \) be the formula

\[
\left( \exists z_1, \ldots, z_n \right) \left( \bigwedge_{i<j \leq n} (w \sim_2 z_i \land z_i \sim_1 z_j \land \left( \bigwedge_{k \in \omega} \mathcal{D}_k(z_i) \right) ) \right)
\]

The formula \( \mathcal{D}_{2n}(w) \) then says there are at least \( n \) elements that are \( \sim_2 \)-equivalent to \( w \) but not pairwise \( \sim_1 \)-equivalent; further, for each finite number \( k \), there are at least \( k \) distinct elements that are \( \sim_1 \)-equivalent to each of the \( n \) elements \( \sim_2 \)-equivalent to \( w \). Put more succinctly, \( \mathcal{D}_{2n}(w) \) states that there are at least \( n \) infinite \( \sim_1 \)-classes in the same \( \sim_2 \)-class as \( w \). Each
$D_1k(z_i)$ is a $\Sigma^0_1$ formula. The infinite conjunction of the $D_1$'s is thus a $\Pi^0_2$ formula. Therefore, $D_2n(w)$ is a $\Sigma^0_3$ formula.

Since $\lceil y \rceil_1 > \lceil x \rceil_1$, $A_2 \models D_2\lceil y \rceil_1$, but $A_2 \not\models D_2\lceil x \rceil_1$. So, $D_2\lceil y \rceil_1$ is a $\Sigma^0_3$ formula true of $\overline{y}$ in $A_2$ but false of $\overline{x}$ in $A_2$. By Proposition 1.2.1 this is a contradiction.

$\Leftarrow$ Assume that the three conditions apply to $\overline{x}$ and $\overline{y}$, and let $\Phi(\overline{z})$ be a $\Sigma^0_3$ formula true of $\overline{y}$. $\Phi$ has the form $\forall_i (\exists u_i)(\phi_i(\overline{z}, u_i))$, where each $\phi_i$ is a $\Pi^0_2$ formula. Let $I$ be the index of a true disjunct, and $\overline{a}$ a witness for $\overline{a}$. As in the proof of Claim 2 one can show $\phi_I(\overline{y}, \overline{a})$ implies $\phi_I(\overline{x}, \overline{a}')$ for some $\overline{a}'$ by showing that $\overline{y}, \overline{a} \leq_2 \overline{x}, \overline{a}'$. As before, this follows from Claim 2.

That $\lceil y \rceil_1 = \lceil x \rceil_1$ for each $y_i \in \overline{y}$ and each $x_i \in \overline{x}$ is part of the assumption. We need to show that, for any possible $\overline{a}$, there exists $\overline{a}'$ such that $\overline{y}, \overline{a} \equiv_0 \overline{x}, \overline{a}'$, and that $\lceil a'_i \rceil_1 = \lceil a_i \rceil_1$ for each $a_i \in \overline{a}$ and each $a'_i \in \overline{a}'$.

Finding values for $\overline{a}'$ works much as it did in Claim 2.

1. Sort each $a_i$ by its $\sim_1$-class, so that each $a_i$ in the same $\sim_1$-class is listed together.

2. Deal with the lists containing $a_i$ that are in the same $\sim_1$-class as an element of $\overline{y}$: Find the $\sim_1$-class that contains the corresponding elements of $\overline{x}$. Assign values in $\overline{a}'$ corresponding to those on the list respecting equality/inequality of the pairs of elements from $\overline{a}$, between $\overline{a}$ and $\overline{y}$ as applicable. There will be enough elements to choose from, since $\lceil y \rceil_1 = \lceil x \rceil_1$ for each $y_i \in \overline{y}$ and each $x_i \in \overline{x}$.
3. Deal with the lists containing $a_i$ that are not in the same $\sim_1$-class as any element of $\overline{y}$, but are in the same $\sim_2$-class as an element of $\overline{y}$ or one of the elements of $\overline{a}$ handled in previous steps.

- If $a_i$ is in a finite $\sim_1$-class, choose the $\sim_1$-class of the same size from the $\sim_2$-class in question. Note that the selected $\sim_1$-class will have no $x$’s or other $a'$’s in it. If it had any $x$’s, then $a_i$ would have been dealt with in the previous step. If it had any other $a'$’s in it, then $a_i$ would have appeared on the list of an $a$ that was dealt with previously.

- If $a_i$ is in an infinite $\sim_1$-class, choose an infinite $\sim_1$-class that has no $x$’s or $a'$’s in it. Such a class must exist because $|y_1|_1 \leq |x_1|_1$. If all the infinite $\sim_1$-classes contain an $x$, $a_i$ would be in the same $\sim_1$-class as a $y$, and would thus have been dealt with in the previous step. If all the infinite $\sim_1$-classes without any $x$’s contain an $a'$, $a_i$ would have to have been in the same $\sim_1$-class as the corresponding $a$, and would have been dealt with at the same time.

Either way, assign values in $\overline{a'}$ from the chosen $\sim_1$-class respecting equality/inequality of the pairs of elements from $\overline{a}$.

4. Finally, deal with a list containing an $a_i$ that is not even in the same $\sim_2$-class as any $y$ or previously-assigned $a$. Pick a $\sim_2$-class that contains no $x$’s and has not had any $a'$’s assigned to it yet that has at least as many infinite $\sim_1$-classes as the $\sim_2$-class containing $a_i$. Such a $\sim_2$-class will exist because there are $\sim_2$-classes with an arbitrary number of infinite $\sim_1$-classes. Pick a $\sim_1$-class and assign values in $\overline{a'}$ in the same manner as in the preceding step.
5. Repeat steps 3–5 until all elements of $\overrightarrow{a'}$ are assigned values.

The validity of the various pieces of atomic information are handled in the same steps as they were in the proof of Claim 2. So, $\overrightarrow{y}, \overrightarrow{a} \equiv_0 \overrightarrow{x}, \overrightarrow{a'}$. The $a'$'s were chosen so that $| [a_i']_1 | = | [a_i]_1 |$ for each $a_i \in \overrightarrow{a}$ and each $a_i' \in \overrightarrow{a'}$. Therefore, the three conditions listed above are sufficient to conclude $\overrightarrow{x} \leq_3 \overrightarrow{y}$.

\[ \square \]

4.2.1.5 $\leq_4$

**Claim 4.** For any tuples $\overrightarrow{x}$ and $\overrightarrow{y}$ of the same length of elements in $A_2$, $\overrightarrow{x} \leq_4 \overrightarrow{y}$ iff all of the following conditions hold:

1. $\overrightarrow{x} \equiv_0 \overrightarrow{y}$
2. $| [y_i]_1 | = | [x_i]_1 |$ for each $y_i \in \overrightarrow{y}$ and each $x_i \in \overrightarrow{x}$.
3. $[y_i]_1 = [x_i]_1$ for each $y_i \in \overrightarrow{y}$ and each $x_i \in \overrightarrow{x}$.

**Proof.** ($\Rightarrow$) Since $\overrightarrow{x} \leq_4 \overrightarrow{y}$ implies $\overrightarrow{x} \leq_3 \overrightarrow{y}$, Claim 3 above gives us $\overrightarrow{x} \equiv_0 \overrightarrow{y}$, $| [y_i]_1 | = | [x_i]_1 |$ for each $y_i \in \overrightarrow{y}$ and each $x_i \in \overrightarrow{x}$, and $[y_i]_1 \leq [x_i]_1$ for each $y_i \in \overrightarrow{y}$ and each $x_i \in \overrightarrow{x}$. Since $\overrightarrow{x} \leq_4 \overrightarrow{y}$ also implies $\overrightarrow{y} \leq_3 \overrightarrow{x}$, Claim 3 also gives us $[x_i]_1 \leq [y_i]_1$ for each $y_i \in \overrightarrow{y}$ and each $x_i \in \overrightarrow{x}$.

($\Leftarrow$) Let $\Phi(\overrightarrow{z})$ be a $\Sigma^0_4$ formula true of $\overrightarrow{y}$. $\Phi$ is of the form $\bigwedge_i (\exists \overrightarrow{u}_i) \phi_i(\overrightarrow{z}, \overrightarrow{u}_i)$, where each $\phi_i$ is a $\Pi^0_3$ formula. Let $I$ be an index for a disjunct true of $\overrightarrow{y}$ and $\overrightarrow{a}$ a witness for the existential quantifier. The equivalence of the first and third parts of Proposition 1.2.1 mean that one can show $\phi_I(\overrightarrow{y}, \overrightarrow{a})$ implies $\phi_I(\overrightarrow{x}, \overrightarrow{a'})$ for some $a'$ by showing that $\overrightarrow{y}, \overrightarrow{a} \leq_3 \overrightarrow{x}, \overrightarrow{a'}$. This can be obtained from Claim 3.
The three conditions of Claim 3 are satisfied for $\vec{y}$ and $\vec{x}$ by assumption.

We need to show that, for any possible $\vec{a}$, there exists $\vec{a}'$ such that $\vec{y}, \vec{a}' \equiv_0 \vec{x}, \vec{a}'$ and $|a_i|_1 \geq |a'_i|_1$ for each $a_i \in \vec{a}$ and each $a'_i \in \vec{a}'$.

To assign values for $\vec{a}'$,

1. Sort each $a_i$ by its $\sim_1$-class, so that each $a_i$ in the same $\sim_1$-class is listed together.

2. Deal with the lists containing $a_i$ that are in the same $\sim_1$-class as an element of $\vec{y}$: Find the $\sim_1$-class that contains the corresponding elements of $\vec{x}$. Assign values in $\vec{a}'$ corresponding to those on the list respecting equality/inequality of the pairs of elements from $\vec{a}$, between $\vec{a}$ and $\vec{y}$ as applicable. There will be enough elements to choose from, since $|\{y_i\}| = |\{x_i\}|$ for each $y_i \in \vec{y}$ and each $x_i \in \vec{x}$.

3. Deal with the lists containing $a_i$ that are not in the same $\sim_1$-class as any element of $\vec{y}$, but are in the same $\sim_2$-class as an element of $\vec{y}$ or one of the elements of $\vec{a}$ handled in previous steps.

   - If $a_i$ is in a finite $\sim_1$-class, choose the $\sim_1$-class of the same size from the $\sim_2$-class in question. Note that the selected $\sim_1$-class will have no $x$’s or other $a’$’s in it. If it had any $x$’s, then $a_i$ would have been dealt with in the previous step. If it had any other $a’$’s in it, then $a_i$ would have appeared on the list of an $a$ that was dealt with previously.

   - If $a_i$ is in an infinite $\sim_1$-class, choose an infinite $\sim_1$-class that has no $x$’s or $a’$’s in it. Such a class must exist because $|\{y_i\}| = |\{x_i\}|$. If all the infinite $\sim_1$-classes contain an $x$, $a_i$ would be in the same $\sim_1$-class as a $y$, and would thus have been dealt with in the previous
step. If all the infinite ∼₁-classes without any x’s contain an a’, aᵢ would have to have been in the same ∼₁-class as the corresponding a, and would have been dealt with at the same time.

Either way, assign values in ⃗a' from the chosen ∼₁-class respecting equality/inequality of the pairs of elements from ⃗a.

4. Finally, deal with a list containing an aᵢ that is not even in the same ∼₂-class as any y or previously-assigned a. Pick the ∼₂-class that has the same number of infinite ∼₁-classes as the ∼₂-class containing aᵢ. Pick a ∼₁-class and assign values in ⃗a' in the same manner as in the preceding step.

5. Repeat steps 3–5 until all elements of ⃗a' are assigned values.

As in the previous claims, we can conclude ⃗y, ⃗a ≡₀ ⃗x, ⃗a' from the method used to choose ⃗a’. Likewise, each a’ was chosen so as to ensure that |[aᵢ]₁| = |[a’ᵢ]₁| and [aᵢ]₁ = [a’ᵢ]₁ for each aᵢ ∈ ⃗a and each a’ᵢ ∈ ⃗a’. Therefore, the three conditions listed above are sufficient to conclude ⃗x ≤₄ ⃗y.

4.2.2 The Proof

4.2.2.1 Categoricity of A₂

Before proving that A₂ is not ∆₀³-categorical, it will be nice to know what its level of categoricity it actually attains.

**Theorem 4.2.1.** A₂ is relatively Δ⁴¹-categorical.

**Proof.** Proving that A₂ is relatively Δ⁴¹-categorical can be done by constructing a Scott family to use with Lemma [1.2.2]. Define the formula sₘₙ(x) to be
\(\bigwedge_{i \in \omega} \mathcal{D}_i(x)\) states that, for each \(i \in \omega\), there are at least \(i\) distinct elements \(\sim_1\)-equivalent to \(x\). Because of that, \(s_\infty(x)\) is equivalent to the statement “\(x\) is in an infinite \(\sim_1\)-equivalence class.” Since each \(\mathcal{D}_i\) is a \(\Sigma^0_1\) formula, \(s_\infty\) is a \(\Pi^0_2\) formula.

Let \(c_n(x)\) be the formula

\[
\left( \exists y_1, \ldots, y_n \right) \left( \bigwedge_{i \leq n} y_i \sim_2 x \land y_i \sim_1 y_j \land \left( \forall z \left( z \sim_2 x \rightarrow \left( \bigvee_{k \leq n} z \sim_1 y_k \lor \neg s_\infty(z) \right) \right) \right) \right)
\]

\(c_n(x)\) states that \([x]_1 = n\). Since \(s_\infty(z)\) is a \(\Pi^0_2\) formula, \(\neg s_\infty(z)\) is a \(\Sigma^0_2\) formula. The part of the formula that starts with \(\forall z\) is thus a \(\Pi^0_3\) formula. Therefore, the whole of \(c_n(x)\) is a \(\Sigma^0_4\) formula.

For any tuple \(\overrightarrow{x}\), let \(S(\overrightarrow{x})\) be the set of formulas

\[
\left\{ \bigwedge_{i \leq [x]} c_{n_i}(x_i) \land s_{m_i}(x_i) \ \middle| \ n_i \in \omega, m_i \in \omega \cup \{\infty\} \right\}
\]

As in Theorem 3.2.1, \(S\) is basically the set of all formulas that say “\(x\) is in a \(\sim_1\)-equivalence class with \(m\) elements, in a \(\sim_2\)-equivalence class that contains \(n\) infinite \(\sim_1\)-equivalence classes.” Each formula in \(S\) is \(\Sigma^0_1\).

For any tuple \(\overrightarrow{x}\), let \(A(\overrightarrow{x})\) be the set of all formulas

\[
\left\{ \bigwedge_{i < j \leq [x]} R_{i,j}(x_i, x_j) \ \middle| \ R_{i,j} \in \{=, \neq, \sim_1, \sim_2\} \right\}
\]

Again, \(A\) is basically the set of all formulas that talk about how individual elements of a tuple are related to each other. Each formula in \(A\) is \(\Sigma^0_0\).
Let Φ be the set of formulas

\[ \{ a(\overline{x}) \land s(\overline{x}) | a \in \mathcal{A}(\overline{x}); s \in \mathcal{S}(\overline{x}) \} \]

Each formula in Φ places the elements of \( \overline{x} \) in a specific \( \sim_1 \)-class within a specific \( \sim_2 \)-class, as well as sorting out the relations between all individual elements of the tuple. So, Φ is a Scott family for \( \mathcal{A}_2 \). There are some formulas which are not satisfied by any element of \( \mathcal{A}_2 \), but this is still acceptable in the definition of Scott family. Each formula in Φ is a \( \Sigma^0_4 \) formula, so Φ is a formally \( \Sigma^0_4 \) Scott family. Therefore, by Theorem 1.2.2, \( \mathcal{A}_2 \) is relatively \( \Delta^0_1 \)-categorical.

4.2.2.2 Not \( \Delta^0_3 \)-Categorical

**Theorem 4.2.2.** \( \mathcal{A}_2 \) is not \( \Delta^0_3 \)-categorical.

**Proof.** Proving that \( \mathcal{A}_2 \) is not \( \Delta^0_3 \)-categorical can be done by using Theorem 1.2.3

- To prove that \( \mathcal{A}_2 \) is 3-friendly, note that Claim 3 shows that \( \leq_3 \) is equivalent to conditions that can be expressed as a c.e. disjunction of first-order sentences expanded to include the quantifier \( \exists^\infty \).

Define the formula \( s'_\infty(x) \) to be \( \exists^\infty z \langle x \sim_1 z \rangle \). Define the formula \( c'_n(x) \) with the same definition as \( c_n(x) \), except replacing \( s_\infty \) with \( s'_\infty \). Define the formula \( D'_n(x) \) as

\[
\left( \exists z_1, \ldots, z_n \right) \left( \bigwedge_{i<j \leq n} \left( x \sim_2 z_i \land z_i \sim_1 z_j \land \left( \exists^\infty k \langle k \sim_1 z_i \rangle \right) \right) \right)
\]

The prime formulas have the same meaning as the non-prime versions, except
with the infinite conjunctions replaced with \( \exists^\infty \). Consider the disjunction
\[
\bigwedge_{j < \omega} \left( \bigvee_{i \in \omega} \left( \left[ c'_j(y_i) \rightarrow D2'_j(x_i) \right] \land \left[ s_n(x_i) \leftrightarrow s_n(y_i) \right] \land \left[ a(\overline{x}, \overline{y}) \right] \right) \right)
\]
The first implication makes sure that condition 3 of Claim 3 is satisfied.
The second implication makes sure that condition 2 of Claim 3 is satisfied.
Finally, the formula from \( A(\overline{x}, \overline{y}) \) makes sure that condition 1 of Claim 3 is satisfied. So, the disjunction is equivalent to the formula \( \overline{x} \leq_3 \overline{y} \).
The disjunction is c.e. which means by Theorem 1.2.9, \( \leq_3 \) is computably enumerable. Therefore, \( A_2 \) is 3-friendly.

- To prove that for each \( \overline{c} \), there is a \( \overline{a} \) that is 3-categorically free over \( \overline{c} \), let \( \overline{c} \) be a tuple from \( A_2 \). Showing freeness requires that, for the chosen \( \overline{a} \), it must be the case that for all \( \overline{b} \), there exist \( \overline{a}' \) and \( \overline{b}' \) such that \( \overline{c}, \overline{a}, \overline{b} \leq_2 \overline{c}', \overline{a}', \overline{b}' \), but \( \overline{c}, \overline{a}' \not\leq_3 \overline{c}', \overline{a} \). This is equivalent to satisfying the following four conditions.

1. \( \overline{c}, \overline{a}, \overline{b} \equiv_0 \overline{c}', \overline{a}', \overline{b}' \)
2. \( |[a_i]_1| = |[a'_i]_1| \)
3. \( |[b_i]_1| = |[b'_i]_1| \)
4. There is some \( I \) such that \( |a_I| > |a'_I| \)

The first three conditions come from Claim 2, the other hypotheses for Claim 2 are automatically satisfied. The fourth condition comes from Claim 3, the other hypotheses for Claim 3 are impossible to contradict.

To choose the \( \overline{a} \) that will be 3-categorically free over \( \overline{c} \), find a \( \sim_2 \) equivalence class that contains more infinite equivalence classes than \( |c|_1 + 3 \) for any
of the $c$'s in $\overrightarrow{c}$. Such a class is guaranteed to exist, since $\{\overrightarrow{c}\}$ is finite and the number of $\sim_2$-classes is not. Choose $a$ to be in the smallest $\sim_1$-equivalence class in there. To choose $a'$, find the $\sim_2$ equivalence class that has $\{a\}_1 - 1$ infinite $\sim_1$-classes. Choose $a'$ to be in the smallest $\sim_1$-equivalence class in there. To choose $\overrightarrow{b}'$, for each $j \leq \{\overrightarrow{b}\}$,

- If $[b_j]_1 < [a']_1$, pick $b'_j$ to be equal to $b_j$.

- If $[b_j]_1 \geq [a']_1$, pick $b'_j$ to be in the corresponding $\sim_1$-class in the $\sim_2$ equivalence class with $[b_j]_1 - 1$ infinite $\sim_1$-classes.

  * If $b_j = a$, make sure $b'_j = a'$; if $b_j \neq a$, make sure $b'_j \neq a'$. This is not an issue, since $|[a]_1| = |[a']_1|$.

With this scheme, the second, third, and fourth conditions listed above are obviously true. Proving the first condition comes down to checking all possible atomic sentences.

- Sentences of the form $c_i R a$ will never be true for either $a$ or $a'$, since both were chosen to be in a different $\sim_2$-class than any $c$.

- Sentences of the form $c_i R b_j$

  * If this sentence is true, then $[b_j]_1 < [a]_1$, so $b'_j = b_j$, and thus $c_i R b'_j$.

  * If this sentence is false, either $b'_j = b_j$ or $b'_j$ is in a $\sim_2$-equivalence class that contains no $c$'s; either way, $\neg(c_i R b'_j)$.

- Sentences of the form $a R b_j$

  $a = b_j$ The algorithm for choosing $\overrightarrow{b}'$ makes sure that $a' = b'_j$ will have the same truth value.
\( a \sim_1 b \) \( b'_j \) will be in the same \( \sim_1 \)-class as \( a' \) iff \( b_j \) was in the same \( \sim_1 \)-class as \( a \).

\( a \sim_2 b \) If this sentence is true, then \( b'_j \) will be in \( [a']_2 \). If this sentence is false, then \( b'_j \) won’t be in \( [a']_2 \).

- Proving that \( \not\leq_3 \) is computably enumerable works like the proof that \( A_2 \) is 3-friendly. Consider the following formula.

\[
\bigvee_{j<\omega} \left( \bigvee_{i \leq \{x\}} \left[ \ell'_j(y_i) \rightarrow \bigvee_{k<j} \ell'_k(x_i) \right] \right) \lor \left[ \neg s_n(x_i) \leftrightarrow s_n(y_i) \right] \lor \left[ \bigwedge_{a \in A}(\neg a(\overline{x}, \overline{y})) \right]
\]

The first implication makes condition 3 of Claim 3 unsatisfied. The second implication makes condition 2 of Claim 3 unsatisfied. Finally, the conjunction of formulas from \( A(\overline{x}, \overline{y}) \) makes condition 1 of Claim 3 unsatisfied.

So, the disjunction is equivalent to the formula \( \overline{x} \not\leq_3 \overline{y} \). The disjunction is c.e. which means by Theorem 1.2.9, \( \not\leq_3 \) is computably enumerable.

Thus, the three hypotheses of Theorem 1.2.3 are true of \( A_2 \). Therefore, there is a computable \( B \) isomorphic to \( A_2 \) for which there is no \( \Delta^0_3 \) isomorphism from \( A_2 \) to \( B \), so \( A_2 \) is not \( \Delta^0_3 \)-categorical. \[\square\]
5.1 Future Work

5.1.1 $J$-Limits for Transfinite Values of $J$

In dealing with $J$-limits in ordinals, this dissertation only considered finite values for $J$ in ordinals less than $\omega^\omega$. This is because the dissertation is concerned with automatic structures, and these are the only values where automatic copies exist. However, it may be worth examining whether or not there are definitions of $J$-limits where $J$ is not finite, and whether or not the pattern of “$\Pi^0_2$J definable but not $\Sigma^0_2$J” continues to hold for computable ordinals.

5.1.2 Even More Complex Nested Equivalence Structures

It should be possible to create automatic nested equivalence structures with more relations and higher levels of categoricity. Here’s a rough outline for how to make $\mathcal{A}_3$, a nested equivalence structure with three relations that is probably not $\Delta^0_4$-categorical. Extend $\mathcal{A}_2$ to include a $\sim_2$-class with infinitely many infinite $\sim_1$-classes, and make the extended structure a single $\sim_3$-class. The next $\sim_3$-class would have two $\sim_2$-classes with infinitely many infinite $\sim_1$-classes, then the next one would have three, and so forth. The proofs for the back-and-forth relations would probably look similar to the proofs used here, and the constructed machines
would be similar as well. Even if it turns out that the structure is not automatic, it should be more than $\Delta_0^n$-categorical. Even if $A_3$ is $\Delta_0^n$-categorical, there should be a way to make nested equivalence structures that aren’t.

5.1.3 General Nested Equivalence Structures

There are general theorems for determining the less than relations for computable linear orderings, and general theorems proving the categoricity of equivalence structures. There doesn’t seem to be any work providing either of those types of theorems for nested equivalence relations.

The definition of a nested equivalence structure given here can also be generalized. This definition assumes that the equivalence relations will be indexed by a finite linear ordering or $\omega$. However, there is no reason one cannot have a structure with equivalence relations that still obey $i < j \rightarrow (\forall x \forall y (x \sim_i y \rightarrow x \sim_j y))$ where the indices are from a more complicated linear ordering, such as $\omega^2$ or $\omega + \omega^*$. It would be interesting to construct a computable nested equivalence structure where no two equivalence relations are equal, and the indices of the equivalence relations has the order type of the rational numbers.

5.1.4 Refining Classifications

The problems covered in this dissertation have been ranked as being “$\Pi_n^0$ but not $\Sigma_n^0$,” or “$\Delta_n^0$ but not $\Delta_{n-1}^0$.” Although this provides a fairly narrow range for the complexity of the problems, there are still finer classifications of the complexity of sets. For example, Ershov’s hierarchy provides a finer ranking of the $\Delta_0^0$ sets. The problems could be pursued even further and shown to be complete at a certain level, thus narrowing down the complexity of the problem to a single Turing degree.
5.2 Other Work by the Author

5.2.1 Background

It is possible to compare the complexity of general computable structures amongst each other by considering the atomic diagrams of the structures. This allows the complexity of various problems to be ranked according to the complexity of the characteristic function of the atomic diagram. For a structure \( \mathcal{A} \), the number \( e \) is called a \textit{computable index} for \( \mathcal{A} \) if \( \varphi_e \) is the characteristic function of the atomic diagram of \( \mathcal{A} \). Sets of computable indices make for natural examples of sets at various levels in the arithmetical hierarchy.

The \textit{index set} of a structure \( \mathcal{A} \) is the set of computable indices for \( \mathcal{A} \). Conversely, given a computable index \( e \), \( \mathcal{A}_e \) is the structure for which \( e \) is a computable index. If \( K \) is a class of structures closed under isomorphism, one can also define the \textit{index set} of \( K \) to be the set of computable indices for structures in \( K \).

There are some classes \( K \) where the index set is so complicated that the complexity of other interesting problems is overwhelmed. In these cases, the notion of \textit{complexity within a larger set} is useful. For a complexity class \( \Gamma \) and sets \( A \subseteq B \), \( A \) is said to be \( \Gamma \) within \( B \) iff there is a set \( C \) such that \( C \) is \( \Gamma \) and \( A = B \cap C \).

5.2.2 The Embedding Problem

Let \( K \) be a class of structures closed under isomorphism. The \textit{computable isomorphism problem} for \( K \) is the set of ordered pairs \( (a, b) \) such that \( a \) and \( b \) are both computable indices for structures in \( K \) and \( \mathcal{A}_a \cong \mathcal{A}_b \). The \textit{computable embedding problem} for \( K \) is the set of ordered pairs \( (a, b) \) such that \( a \) and \( b \) are both computable indices for structures in \( K \) and where there exists an embedding of \( \mathcal{A}_a \) into \( \mathcal{A}_b \). Although these notions are related, the complexity of the computable
isomorphism problem can be strictly greater or strictly less than the complexity of the computable embedding problem. The author of this dissertation suggested the case where \( K \) is the class of free groups on two or more generators as an example where the computable isomorphism problem is strictly more complicated than the computable embedding problem.

A free group with two or more generators can be embedded into any other free group with two or more generators, so the computable embedding problem for \( K \) is trivial within \( K \). By contrast, in \([4]\), the isomorphism problem is shown to be m-complete \( \Delta^0_3 \) within \( K \).

5.2.3 Describing Free Groups

In \([5]\), the authors worked on pinpointing the complexity of other problems involving classes of free groups. For example, the complexity of the index set for the class of all free groups is shown to be m-complete \( \Pi^0_4 \) within the class of all groups. The author of this dissertation assisted analysis by writing a computer script to test properties of group elements, so that the working group would know more quickly if their conjectures had counterexamples.
References


