Computability-Theoretic Properties of Partial Injections, Trees, and Nested Equivalences

by Leah Marshall

B.A. in Mathematics and Statistics, June 2004, Northwestern University
M.A. in Mathematics, May 2010, The George Washington University

A Dissertation submitted to

The Faculty of
The Columbian College of Arts and Sciences
of The George Washington University
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

August 31, 2015

Dissertation directed by

Valentina Harizanov
Professor of Mathematics

The Columbian College of Arts and Sciences of The George Washington University certifies that Leah Marshall has passed the Final Examination for the degree of Doctor of Philosophy as of May 12th, 2015. This is the final and approved form of the dissertation.

Computability-Theoretic Properties of
Partial Injections, Trees, and Nested Equivalences

Leah Marshall

Dissertation Research Committee:

Valentina Harizanov, Professor of Mathematics, Dissertation Director

Russell Miller, Professor of Mathematics, CUNY Graduate Center, Committee Member

Jozef Przytycki, Professor of Mathematics, Committee Member

Acknowledgments

Above all, I would like to thank my advisor, Valentina Harizanov for her guidance, support, and never-ending faith in me and my abilities. I would also like to thank Russell Miller for the motivation behind examining nested equivalence structures, for his help reading countless drafts, and for his constant encouragement and frequent trips to DC. I would like to thank Jennifer Chubb Reimann, who came up with the idea to examine a correspondence between nested equivalence structures and trees in the first place.

I would like to thank all my friends and family, especially Jason, Chester, and my parents, for their continued love and support through the years. Also to my fellow classmates and officemates along the way, especially Tyler, Joe, Erblin, Hakim, Trang, and Iva — they always offered a patient ear and a helping hand in the office when I could not myself make it in.

My greatest appreciation goes out to my dissertation committee, including Jozef Przytycki, Michele Friend, E. Arthur (Robbie) Robinson, and Murli Gupta. And additionally, to the George Washington University's Summer Program for Women in Mathematics; I wouldn't have ended up here without it.

This dissertation was supported in part by NSF grant DMS-1202328, which I gratefully acknowledge for assistance with books and travel, both of which helped greatly in my research efforts.

The George Washington University Leah Marshall

Date of Defense May 12th, 2015

Abstract

Computability-Theoretic Properties of
Partial Injections, Trees, and Nested Equivalences

We examine computable isomorphisms and other related computability-theoretic properties of three types of countable structures – partial injection structures, full finite height trees, and nested equivalence structures.

We generalize the notion of injection structures – consisting of a set of natural numbers and an injective (1-1) function – and define partial injection structures by considering only partial functions. We make significant progress towards completely classifying computable categoricity for partial injection structures by giving two sufficient conditions for relative computable categoricity. We furthermore explore the other conditions on partial injection structures, and show non-computable categoricity. We do the same for Δ^0_2-categoricity, giving sufficient conditions for relative Δ^0_2-categoricity, and exploring many of the other conditions to show non-Δ^0_2-categoricity. Finally, we show that all partial computable injection structures must be relatively Δ^0_3-categorical.

We next explore finitely nested equivalence structures and show that they can be represented as trees. Furthermore, we show that every tree under certain conditions, specifically being full and of finite height ≥ 2, can be represented as a nested equivalence structure. Moreover, we do this in an algorithmic manner and build a functor between the two types of structures. We show that this functor behaves nicely – it is full, faithful, essentially onto, and computable. We then leverage the ability to go between trees and nested equivalence structures, and transfer results between the two. We provide previously unknown answers to questions related to computable categoricity and the Turing degree spectra of nested equivalence structures.

Contents

List of Figures

List of Symbols

$\prec_T$	Order relation on tree $\mathcal{T}$
$\leq_T$	Turing reducible
$\simeq$	Isomorphic
$\equiv_T$	Turing equivalent
$\simeq_c$	Computably isomorphic
$\simeq_{\mathbf{d}}$	**d**-isomorphic
$\simeq_{\Delta_n}$	Δ^0_n-isomorphic
$\oplus$	Computable join
$\emptyset'$	Turing jump of the empty set; the Halting set
$\emptyset^{(n)}$	The nth Turing jump of the empty set
$\mathbf{0}'$	Turing degree of the jump of the empty set; Turing degree of the Halting set
$\mathbf{0^{(n)}}$	Turing degree of the nth jump of the empty set
$\mathbf{a}$	Turing degree of set A
$\mathbf{a}'$	Turing degree of set A'
$\mathbf{a^{(n)}}$	Turing degree of set $A^{(n)}$
$[a]_E$	Equivalence class of a under equivalence relation E
$\mathcal{A}$	Countable mathematical structure.
A'	Turing jump of set A

$A^{(n)}$	nth Turing jump of set A
(A, E)	Equivalence structure with universe A and equivalence relation E
$(A, E_1, \ldots, E_n)$	Nested equivalence structure with universe A and equivalence relations $E_1, \ldots, E_n$
(A, f)	Injection or partial injection structure with universe A and 1-1 function f
$\mathcal{A}_{\mathcal{T}_A}$	Nested equivalence structure built out of tree $\mathcal{T}$ using universe $A \subseteq \mathbb{N}$
$\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}$	Nested equivalence structure built out of tree $\mathcal{T}$ using universe $\mathbb{N}$
$\mathscr{C}_A$	Partial characteristic function of set A
$\mathscr{C}_R$	Partial characteristic function of relation R
$C_{\mathcal{A}}$	Equivalence classes of nested equivalence structure $\mathcal{A}$, without repetition
$\mathrm{card}(A)$	Cardinality of set A
$\mathrm{CatSpec}(\mathcal{A})$	Categoricity spectrum of structure $\mathcal{A}$
χ_A	Characteristic function of set A
χ_R	Characteristic function of relation R
$\mathbf{d}$	Turing degree
D_1^0	Difference of two c.e. sets
D_n^0	Difference of two Σ_n^0 sets
$\deg(A)$	Turing degree of set A
$\deg(\mathcal{A})$	Turing degree of structure $\mathcal{A}$
Δ_1^0	Level 1 in the arithmetical hierarchy, $\Sigma_1^0 \cap \Pi_1^0$
Δ_n^0	Level n in the arithmetical hierarchy, $\Sigma_n^0 \cap \Pi_n^0$
Δ_n^A	Level n in the relativized arithmetical hierarchy, $\Sigma_n^0 \cap \Pi_n^0$ relative to set A
$\mathrm{DgSp}(\mathcal{A})$	Turing degree spectrum of structure $\mathcal{A}$
$\mathrm{DgSp}_{\mathcal{A}}(R)$	Turing degree spectrum of relation R on structure $\mathcal{A}$
$\mathrm{dom}(f)$	Domain of function f

FFT	Category of full, finite height trees
f_t	Function f computed with timebound t
$f(x)\downarrow$	Function f halts on input x
$f(x)\uparrow$	Function f with input x does not halt, or computes forever
$\mathrm{Hom}_{\mathbf{C}}(A, B)$	Set of all homomorphisms in category $\mathbf{C}$ between objects A and B
$\mathrm{ht}(\mathcal{T})$	Height of tree $\mathcal{T}$
K	Halting set
$\mathrm{level}_{\mathcal{T}}(x)$	The level of node x on tree $\mathcal{T}$
NEquiv	Category of finitely nested equivalence structures
$\mathcal{O}_f(a)$	Orbit of a under f
$\varphi_0, \varphi_1, \varphi_2, \ldots$	Algorithmic list of all partial computable functions
Π^0_1	Level 1 in the arithmetical hierarchy, $\forall x \left(\cdot \right)$
Π^0_n	Level n in the arithmetical hierarchy, $\forall x_1 \exists x_2 \forall x_3 \cdots x_n \left(\cdot \right)$
Π^A_n	Level n in the relativized arithmetical hierarchy, Π^A_n relative to set A
PInj	Set of indices of partial injection structures
$p(\mathcal{T}, x, i)$	ith level predecessor of node x on tree $\mathcal{T}$
P_x	Set of all predecessors of a node x on a given tree
$\mathrm{range}(f)$	Range of function f
Σ^0_1	Level 1 in the arithmetical hierarchy, $\exists x \left(\cdot \right)$
Σ^0_n	Level n in the arithmetical hierarchy, $\exists x_1 \forall x_2 \exists x_3 \cdots x_n \left(\cdot \right)$
Σ^A_n	Level n in the relativized arithmetical hierarchy, Σ^0_n relative to set A
$\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}$	Tree built out of nested equivalence structure $\mathcal{A}$ using universe $\mathbb{N}$
$\mathcal{T}_{\mathcal{A}_T}$	Tree built out of nested equivalence structure $\mathcal{A}$ using universe $T \subseteq \mathbb{N}$
$\mathcal{T} = (T, \prec)$	Tree with universe T and partial order $\prec$
W_e	Domain of φ_e; the eth c.e. set

Chapter 1

Introduction

Computability theory formalizes the notions of what computers can do. Informally, we say that a mathematical object is *computable* if there is some computer program or effective algorithm which computes it. Although mathematicians have been studying algorithms for centuries, the formal study of computability theory is a relatively new one, most commonly thought to have begun in 1936 with a paper from Alan Turing. There are various models of computability theory which fully define and standardize these notions of computability, including: recursive functions due to Gödel and Kleene, Turing computability due to Alan Turing, and unlimited register machine computability due to Sturgis and Shepherdson in the 1950s. Any of these notions are equivalent. In fact, Church-Turing's thesis states that any intuitively computable function – that is, any function for which we can describe in words some finite algorithm – is formally computable by any of these above notions of computability.

This dissertation focuses on computable structures, especially on the computability-theoretic categoricity of partial injection structures and nested equivalence structures. We further develop the notions of computability and effective categoricity in the first two sections of this chapter. In our exploration of nested equivalence structures in later chapters, we rely heavily on ideas from category theory, therefore the final section of this chapter discusses the basic notions of category theory.

1.1 Computability

We take the natural numbers, $\mathbb{N} = \{0, 1, 2, 3, \ldots\}$, as the universe of infinite structures in computability theory. As is standard, throughout this dissertation when we say, for example, "$x > 1$",

we really mean "$x \in \mathbb{N}$ and $x > 1$". Additionally, when we say "infinite" we mean "countably infinite". Following conventions of the field, we also often use ω to refer to the set of natural numbers. We now give a basic overview of the notions of computability. For a more thorough discussion of computability theory, see [42] and [14].

The study of computability theory often begins with the Turing machine, which can be thought of as the mathematical idealization of a computer. It was devised in 1936 by Alan Turing as a theoretical device which consists of "ticker tape"-style registers and a tapehead that can move among the registers and can read, write, and erase what's in each register. The Turing machine was never meant to be an actual device, but simply a thought experiment to mimic the steps that a human could do with a pencil and paper (read, write, erase, focus his attention on a new part of the paper). Nevertheless, people have since built actual functioning Turing machine devices and brought this thought experiment into a reality. Furthermore, Turing proved that such a device could indeed carry out any mathematical computation that can be described as an algorithm, even the complex and lengthy computations that are possible with modern day computers.

A Turing machine program consists of a finite list of instructions for the various registers (read, write, erase, move). Since each program is a finite list of instructions, and each instruction can be one of only finitely many types, there are countably many Turing machine programs. Furthermore, we can encode each of these instructions, and hence each of these programs, so that we can list out all Turing machine programs in an effective (that is, algorithmic) manner: $P_0, P_1, P_2, \ldots$

Following standard conventions of how to input natural numbers into the Turing machine and how to read the output from a Turing machine, each program with n inputs can be thought of as computing some n-ary function. Therefore, we define an n-ary function on the natural numbers, $f : \mathbb{N}^n \to \mathbb{N}$, to be **computable** if we can write a Turing machine program that computes it. More specifically, if this Turing machine program takes as input any $x_1, \ldots, x_n \in \mathbb{N}$ and outputs $f(x_1, \ldots, x_n)$ in some finite number of steps, then the function is said to be **total computable**, or simply **computable**. If the algorithm is allowed to possibly compute forever (that is, take an infinite number of steps), then f is said to be **partial computable**. If the function f does indeed compute forever on some input $(x_1, \ldots, x_n)$, then we write $f(x_1, \ldots, x_n) \uparrow$ and say that $(x_1, \ldots, x_n) \notin \text{dom}(f)$. Otherwise, we write $f(x_1, \ldots, x_n) \downarrow$, or $(x_1, \ldots, x_n) \in \text{dom}(f)$. Clearly all total computable functions are also partial computable, though the reverse is not necessarily true.

Since we can list out Turing machine programs in an algorithmic manner, we can list out all their corresponding n-ary functions by letting $\varphi_e^{(n)}$ represent the n-ary function that results from running Turing machine program P_e with n inputs. In this way we can effectively list all partial computable

n-ary functions: $\varphi_0^{(n)}$, $\varphi_1^{(n)}$, $\varphi_2^{(n)}$, ...

We can now define computability for sets and relations in the following way. Let A be a set of natural numbers, let R be some n-ary relation on $\mathbb{N}$, and define the characteristic functions of A and R, χ_A and χ_R respectively, as follows:

$$\chi_A(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{if } x \notin A \end{cases}, \quad \chi_R(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } R(x_1, \ldots, x_n) \text{ is true} \\ 0 & \text{if } R(x_1, \ldots, x_n) \text{ is not true} \end{cases}$$

Then we say that A is a **computable set** if its characteristic function is computable, and we say that R is a **computable relation** if its characteristic function is computable.

Similarly, we can define the partial characteristic functions of a set A or a relation R as follows:

$$\mathscr{C}_A(x) = \begin{cases} 1 & \text{if } x \in A \\ \uparrow & \text{if } x \notin A \end{cases}, \quad \mathscr{C}_R(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } R(x_1, \ldots, x_n) \text{ is true} \\ \uparrow & \text{if } R(x_1, \ldots, x_n) \text{ is not true} \end{cases}$$

We say that A or R is **computably enumerable**, abbreviated by **c.e.**, if its partial characteristic function is computable. Although being c.e. does not necessarily imply being computable, if we place a timebound, t, on the computation of our algorithm, then our algorithm will always halt in a finite number of steps, namely t, and hence will yield a computable function, set, or relation. We denote a function, set, or relation computed under such a timebound as f_t, A_t, or R_t.

We can extend the notion of the Turing machine to that of an oracle Turing machine, which, in addition to performing the tasks of a Turing machine, can perform the task of consulting an oracle. For our purposes we define an **oracle** to be any set, relation, or function of which we can ask questions about membership, truth value, or output value during the computation of our Turing machine program. This gives rise to the notion of relative computability; given two sets A and B we say that B is **A-computable** (equivalently, B is **computable relative to** A, or B is **Turing reducible** to A, or $B \leq_T A$) if there exists a Turing machine which, if given information about membership in A, can compute B. This definition extends in the natural way to relative computability of functions and relations. We can also extend the notions of partial computability and of encoding Turing machine programs to relative partial computability and encoding oracle Turing machine programs, resulting in an effective list of all A-partially computable functions: φ_0^A, φ_1^A, φ_2^A, ... , called a Gödel numbering. Note that there are other types of reducibility studied in computability theory. For instance, many-one reducibility, or m-reducibility, is a particular stronger

form of Turing reducibility.

We can also define two sets A and B to be **Turing equivalent**, written $A \equiv_T B$, if $A \leq_T B$ and $B \leq_T A$. It is easy to check, then, that Turing equivalence is indeed an equivalence relation and that $\leq_T$ is a partial order (under "$\equiv_T$") on subsets of $\mathbb{N}$. Note that it is not, however, a total order, as there do exist sets A and B such that $A \not\leq_T B$ and $B \not\leq_T A$. Intuitively, Turing reducibility and Turing equivalence give us an idea about the computability-theoretic difficulty of certain problems – whether one problem is less difficult than another ($\leq_T$), equally as difficult as another ($\equiv_T$), or the two are incomparable. Because of this notion of varying levels of "difficulty" of problems, we use the term **Turing degree** to refer to equivalence classes under Turing equivalence. That is, for a set A we define the following:

$$\text{Turing degree of A} \stackrel{\text{def}}{=} \deg(A) = \mathbf{a} = \{B \subseteq \mathbb{N} : B \equiv_T A\}$$

Note that we can think of a relation as a set of tuples under which the relation holds, and we can identify a function with its graph. The Turing degree of a relation or a function can then be defined using the above definition. Problems (sets, functions, or relations) of intuitively the same level of "difficulty" will therefore have the same Turing degree. We say that $\mathbf{a} \leq \mathbf{b}$ for Turing degrees $\mathbf{a} = \deg(A)$ and $\mathbf{b} = \deg(B)$ iff $A \leq_T B$. It is easy to check that $\leq$ forms a partial order on Turing degrees. We use the term **Turing degree hierarchy** to refer to the interplay and relationships of these different levels of difficulty as defined by their Turing degrees, and it can be easily shown that there are uncountably many Turing degrees in the Turing degree hierarchy.

Given two sets A and B, we define the **computable join** of A and B, denoted $A \oplus B$ as:

$$A \oplus B \stackrel{\text{def}}{=} \{2a : a \in A\} \cup \{2b + 1 : b \in B\}$$

It is easy to show that $A \leq_T A \oplus B$ and $B \leq_T A \oplus B$. Furthermore, $\deg(A \oplus B)$ is in fact the unique least upper bound of the Turing degrees of A and B. We often denote this least upper bound by $\deg(A \oplus B) = \deg(A) \cup \deg(B)$, or simply $\mathbf{a} \cup \mathbf{b}$. Since this least upper bound exists for any Turing degrees $\mathbf{a}$ and $\mathbf{b}$, the Turing degrees therefore form an upper semilattice. We should note that the Turing degrees do not form a lower semilattice, as there do exist Turing degrees with no greatest lower bound — so-called "exact pairs".

We consider now which partial computable functions halt, and which do not, given an oracle A.

We define $A' =$ the **jump** (or Turing jump) of a set A as follows.

$$A' = \{e : \varphi_e^A(e) \downarrow\}$$

We denote the Turing degree of A' by $\deg(A') = \mathbf{a}'$. In general we have that $A <_T A'$, and hence $\mathbf{a} < \mathbf{a}'$; this better explains the given terminology, as each time we apply the jump operator to a set, we "jump up" a level of difficulty in the Turing degree hierarchy. We can take multiple jumps of a set, by letting $A'' = (A')'$, $A''' = (A'')'$, and in the general case $A^{(n)} = (A^{(n-1)})'$. If we take A to be the empty set, or indeed any computable set, we get the set $\emptyset' = K =$ the **halting set** $= \{e : \varphi_e(e) \downarrow\}$. Because any computable set can be computed without the use of an oracle, any computable set is $\leq_T$ than any other set. This concept gives rise to the convention that $\deg(\emptyset) = \mathbf{0}$. Therefore $\deg(\emptyset') = \mathbf{0}'$, ..., $\deg(\emptyset'') = \mathbf{0}''$, and in general $\deg(\emptyset^{(n)}) = \mathbf{0^{(n)}}$.

The following defines the **arithmetical hierarchy**. We say that a set of natural numbers X is Σ_n^0 if X can be expressed as: $X = \{x : \exists y_1 \forall y_2 \cdots Q y_n \ R(x, y_1, \ldots, y_n)\}$ and X is Π_n^0 if X can be expressed as: $X = \{x : \forall y_1 \exists y_2 \cdots Q y_n \ R(x, y_1, \ldots, y_n)\}$ for some computable relation, R, where "$\forall$" and "$\exists$" continue to alternate and $Q = \exists$ or $\forall$ depending on whether n is even or odd. We say that a set X is Δ_n^0 if $X \in \Sigma_n^0 \cap \Pi_n^0$. We say that a set X is D_n^0 if X is the difference of two Σ_n^0 sets, or, equivalently if $X =$ the intersection of a Σ_n^0 set and a Π_n^0 set. We say that X is Σ_n^0**-complete** (similarly Π_n^0**-complete** or D_n^0**-complete**), if X is Σ_n^0 (similarly Π_n^0 or D_n^0) and for any Σ_n^0 (similarly Π_n^0 or D_n^0) set Y, Y is m-reducible to X. Equivalently, there exists a total computable function f such that $a \in Y \iff f(a) \in X$.

Post's theorem gives us a connection between the arithmetical hierarchy and the Turing degree hierarchy. Specifically, it states that $\emptyset^{(n)}$ is Σ_n^0-complete (hence $\overline{\emptyset^{(n)}}$ is Π_n^0-complete) for $n > 0$, and A is Δ_{n+1}^0 iff A is $\emptyset^{(n)}$-computable. This gives us that Δ_1^0 sets are exactly computable sets, and Σ_1^0 sets are exactly c.e. sets. Note, we can relativize the arithmetical hierarchy and Post's theorem to discuss sets which are Σ_n^0, Π_n^0, and Δ_n^0 relative to an oracle A, or simply Σ_n^A, Π_n^A, and Δ_n^A.

1.2 Computable Structure Theory

The notion of a structure is quite familiar throughout many areas of mathematics. For instance, linear orderings, trees, graphs, groups, fields, and vector spaces to name a few. Computable structure theory is a branch of mathematics which seeks to investigate the algorithmic nature of these various mathematical structures. For a full and more thorough discussion on the subject, see [3], [18], [24],

[25], [27], and/or [34].

We generally study only **countable structures** – ones in which the universe or domain is a countable set and the constants, relations, and functions which define such a structure are also countable – for computable languages. A **computable language** can be thought of as a tuple of computably presented sets of variable, constant, relation, function, and logical symbols. For the purposes of this dissertation, the languages we consider are always finite. For notation, we tend towards using a script letter to refer to the structure, and the corresponding latin letter to refer to the universe of the structure. For instance, in a linear ordering, $\preceq$, over some set $A \subseteq \mathbb{N}$, we let $\mathcal{A} = (A, \preceq)$ represent the corresponding linearly ordered structure. Loosely defined, then, an **isomorphism** from a structure $\mathcal{A}$ to a structure $\mathcal{B}$ is a 1-1 and onto function $f : A \to B$ such that the associated constants, relations, and functions are preserved. Two structures $\mathcal{A}$ and $\mathcal{B}$ are said to be **isomorphic**, denoted $\mathcal{A} \simeq \mathcal{B}$, if such an isomorphism exists, in which case we sometimes refer to $\mathcal{B}$ as an **isomorphic copy** or simply just a **copy** of $\mathcal{A}$.

Computable structure theory examines the computability-theoretic properties of these mathematical structures. We define a **computable structure** to be a structure in which the universe is computable and the associated relations and functions or operations which define the structure are computable. For example, in the linearly ordered structure given above, $\mathcal{A} = (A, \preceq)$ is computable iff A is a computable set and $\preceq$ is a computable relation. In much of the literature on the subject, when given a (not necessarily computable) structure $\mathcal{A}$, we call a structure $\mathcal{B}$ a **computable copy** of $\mathcal{A}$, or equivalently a **computable presentation** of $\mathcal{A}$, iff $\mathcal{B} \simeq \mathcal{A}$ and $\mathcal{B}$ is computable. We can take these definitions one step further and for a (possibly noncomputable structure) $\mathcal{A}$, we can talk about the Turing degree of the structure. For a countable structure $\mathcal{A}$ with finitely many relations, functions, and constants (which is what we consider in this dissertation), we can define the **Turing degree of a structure** $\mathcal{A}$ to be the least upper bound of the Turing degrees of the relations and functions that define the structure. That is, if $\mathcal{A} = (A, R_1, \ldots, R_n, f_1, \ldots, f_m, c_1, \ldots, c_p)$ then we can define:

$$\deg(\mathcal{A}) \stackrel{\text{def}}{=} \deg(A) \oplus \deg(R_1) \oplus \cdots \oplus \deg(R_n) \oplus \deg(f_1) \oplus \cdots \oplus \deg(f_m)$$

Note that the finitely many constants in our definition of $\mathcal{A}$ do not affect the Turing degree of the structure.

One of the fundamental questions that computable structure theory examines is that of computable isomorphisms. Although isomorphisms preserve the underlying structure of these math-

ematical entities, they do not in fact preserve the computability-theoretic properties. Therefore, from a computability perspective, two isomorphic structures may not in fact be the "same". The situation does not improve even if we require that the structures themselves be computable. That is, it is relatively easy to construct structures $\mathcal{A}$ and $\mathcal{B}$ which are both computable, and in which $\mathcal{A} \simeq \mathcal{B}$, but some property (for example, a set or a relation) computable in $\mathcal{A}$ is not computable in $\mathcal{B}$. This gives rise to the concept of computable isomorphism. For two isomorphic structures, $\mathcal{A}$ and $\mathcal{B}$, we say that $\mathcal{A}$ and $\mathcal{B}$ are **computably isomorphic**, sometimes denoted $\mathcal{A} \simeq_c \mathcal{B}$, iff there is an isomorphism $h : \mathcal{A} \to \mathcal{B}$ which is itself a computable function. Note that the existence of such an h is not guaranteed. It is easy to see, though, that if such a computable isomorphism does exist, then any computable property in $\mathcal{A}$ will also be computable in $\mathcal{B}$ and vice versa, since this isomorphism guarantees an algorithm with which to transfer properties from one structure to the other. We sometimes divide structures up into isomorphism classes, where we think of isomorphism as an equivalence relation on the given type of structures and an **isomorphism type** as simply a class under the "isomorphism" equivalence relation on that structure. We can equally well divide structures up into computable isomorphism classes, where we take "computable isomorphism" as an equivalence relation on all the computable copies of $\mathcal{A}$. That is, we take all the computable copies of some structure $\mathcal{A}$, and examine which of those computable copies are computably isomorphic to each other. A **computable isomorphism type** is then simply an equivalence class under the equivalence relation of "computable isomorphism". The **computable dimension** of a computable structure is defined to be the number of computable isomorphism classes of that structure.

To fully examine the subject of computable isomorphisms, one essential question asked is: given a computable structure, when will all of its computability-theoretic properties always transfer to its computable copies? Or in other words, when will all computable copies of that structure be isomorphic via a computable isomorphism? To this end, we say that a computable structure $\mathcal{A}$ is **computably categorical** iff every computable structure isomorphic to $\mathcal{A}$ is computably isomorphic to $\mathcal{A}$. Note that a structure is computably categorical iff its computable dimension is 1. Another related notion is that of relative computable categoricity, where we extend the concept of computable categoricity, even when our countable structures, $\mathcal{A}$ and $\mathcal{B}$, are not themselves computable. A (possibly non-computable) structure $\mathcal{A}$ is said to be **relatively computably categorical** if for every isomorphic copy $\mathcal{B}$, there is an isomorphism $h : \mathcal{A} \to \mathcal{B}$ such that h is computable in $\mathcal{A} \oplus \mathcal{B}$, or equivalently that is computable in $\deg(\mathcal{A}) \cup \deg(\mathcal{B})$.

We can extend the notions of computable isomorphisms, computable dimension, computable categoricity, and relative computable categoricity to higher levels in the arithmetical hierarchy and

the Turing degree hierarchy. We say that structures $\mathcal{A}$ and $\mathcal{B}$ are $\mathbf{\Delta^0_n}$**-isomorphic**, sometimes denoted $\mathcal{A} \simeq_{\Delta_n} \mathcal{B}$ iff there is an isomorphism $h : \mathcal{A} \to \mathcal{B}$ which is itself a Δ^0_n-function. Given a Turing degree, $\mathbf{d}$, we say that structures $\mathcal{A}$ and $\mathcal{B}$ are **d-isomorphic**, sometimes denoted $\mathcal{A} \simeq_{\mathbf{d}} \mathcal{B}$ iff there is an isomorphism $h : \mathcal{A} \to \mathcal{B}$ which is itself a $\mathbf{d}$-computable function. The **d-computable dimension** of a computable structure is then defined to be the number of $\mathbf{d}$-computable isomorphism classes of that structure. We say that a structure is Δ^0_n**-categorical** (respectively **d-categorical**) iff every computable structure isomorphic to $\mathcal{A}$ is Δ^0_n-isomorphic (respectively $\mathbf{d}$-isomorphic) to $\mathcal{A}$. Note that for $\mathbf{d} = \mathbf{0^{(n)}}$, standard convention is that $\mathbf{0^{(n)}}$-computably categorical structures are generally referred to as Δ^0_{n+1}-categorical. We say that a computable structure $\mathcal{A}$ is **relatively** Δ^0_n**-categorical** if for every isomorphic copy $\mathcal{B}$, there is an isomorphism $h : \mathcal{A} \to \mathcal{B}$ such that h is $\Delta^{\mathcal{B}}_n$, or equivalently, h is computable in $\deg(\mathcal{B})^{(n-1)}$.

One final concept which ties all of these notions together is to examine the entire spectrum or range of Turing degrees for which we have some type of computability-theoretic categoricity. For any computable structure, $\mathcal{A}$, the **categoricity spectrum** of $\mathcal{A}$ is defined as follows:

$$\mathrm{CatSpec}(\mathcal{A}) = \{\mathbf{d} : \mathcal{A} \text{ is } \mathbf{d}\text{-computably categorical}\}$$

Now, for a slightly different approach to examining isomorphisms on countable structures, we use the isomorphisms as a way to see how "difficult" the structure itself is. We can look at the collection of all Turing degrees of isomorphic copies of our structure. If we consider only isomorphic copies whose domain is ω, we can then define the **Turing degree spectrum** of a countable structure $\mathcal{A}$ as follows:

$$\mathrm{DgSp}(\mathcal{A}) = \{\deg(\mathcal{B}) : \mathcal{B} \simeq \mathcal{A}\}$$

Furthermore, we can also discuss the effect that various isomorphisms have on a given relation on our structure. The **Turing degree spectrum of relation** R **on** $\mathcal{A}$ is defined as:

$$\mathrm{DgSp}_{\mathcal{A}}(R) = \{\deg\big(f(R)\big) : \mathcal{B} \simeq \mathcal{A} \text{ via } f \text{ and } \mathcal{B} \text{ is computable}\}$$

If for some Turing degree $\mathbf{d}$, we have that $\mathbf{d} \in \mathrm{DgSp}_{\mathcal{A}}(R)$ (in other words, that $\mathbf{d} = \deg\big(f(R)\big)$ for some isomorphism from $\mathcal{A}$ to a computable copy of $\mathcal{A}$), then we say that $\mathbf{d}$ is **realized** in $\mathrm{DgSp}_{\mathcal{A}}(R)$ via f.

Up until this point, we have mostly discussed ways of examining isomorphisms on a particular structure, $\mathcal{A}$. We wish to also consider isomorphisms on arbitrary structures of a certain type,

however. Roughly speaking, the isomorphism problem is the question of: can we determine, or how difficult is it to determine whether two arbitrary structures are isomorphic? For a given class, $\mathscr{C}$, of computable structures (say computable linear orderings, computable graphs, computable groups, etc), we begin by fixing some enumeration of structures in $\mathscr{C}$. That is, we index the structures in some manner so that for each e the structure $\mathcal{A}_e$ is well-defined with universe $\mathbb{N}$, and such that our enumeration contains all structures in the class. We can now more formally state the **isomorphism problem** for $\mathscr{C}$ as: what is the Turing degree of the following set?

$$\{\langle i, j\rangle : \mathcal{A}_i, \mathcal{A}_j \in \mathscr{C} \text{ and } \mathcal{A}_i \simeq \mathcal{A}_j\}$$

For many classes of structures, this has historically been a difficult problem, and one of great interest in the field of computable structure theory.

1.3 Category Theory

The study of category theory seeks to abstract the notions of functions and systems of functions on various structures, objects, and other mathematical entities. It is a relatively new field of study, generally agreed to have begun in 1945 with a paper by Samuel Eilenberg and Saunders Mac Lane. We give here only a very basic overview of category theory and the concepts needed in this dissertation. For a more detailed examination of category theory, see [4] and [32].

We begin by defining a **category** to be a mathematical structure which consists of all of the following:

- A class of **objects** which we represent with letters A, B, C, ...
- A class of **arrows** (also called **morphisms**) between the objects represented with letters f, g, h, ..., and denoted as $f : A \to B$
- A binary operation on morphisms called **composition** represented by "$\circ$" such that:
  - If $f : A \to B$ and $g : B \to C$, then $g \circ f : A \to C$
  - Associativity holds for $\circ$ — $(h \circ g) \circ f = h \circ (g \circ f)$
  - Identity holds for $\circ$ — there exists a morphism $1_A : A \to A$ such that for $f : A \to B$, $f \circ 1_A = f = 1_B \circ f$.

For a given morphism $f : A \to B$, we call A the **domain** of the morphism f, and we call B the **codomain** of the morphism f.

The simplest example of a category takes the objects to be sets, and the arrows to be functions between the sets. Composition is then defined in the usual way and it is clear that all the necessary properties of being a category are satisfied. While this simplistic example may drive our intuition behind categories, a category is indeed anything that satisfies this particular definition. A slightly less simplistic, but still quite straightforward, example of a category takes the objects to be all partially ordered sets, and the morphisms to be all monotone functions on those sets. It is easy to check that all the necessary properties of being a category are indeed satisfied.

When we define a category, we generally give it a name to refer to later – usually some short word or abbreviation which we put in bold letters and which usually in some way alludes to the pieces involved in the definition of that particular category. For instance, the two examples given above are often denoted as the category **Sets** and the category **Pos**, respectively.

In the language of category theory, we define an isomorphism as an arrow which has an inverse. Formally, in a category $\mathbf{C}$, an arrow $f : A \to B$ is an **isomorphism** if there exists an arrow (we call it f^{-1} since it is easily shown that if such a morphism exists, it is unique) such that $f^{-1} \circ f = 1_A$ and $f \circ f^{-1} = 1_B$. Naturally, in this case A and B are said to be **isomorphic**.

Just as categories deal with mappings between objects, functors examine mappings between categories. A **functor** is a mapping of objects to objects and morphisms to morphisms such that domains, codomains, identity, and composition are all preserved. Specifically, if $\mathbf{C}$ and $\mathbf{D}$ are categories, then $F : \mathbf{C} \to \mathbf{D}$ is a functor iff $F(f : A \to B) = F(f) : F(A) \to F(B)$, $F(1_A) = 1_{F(A)}$, and $F(g \circ f) = F(g) \circ F(f)$.

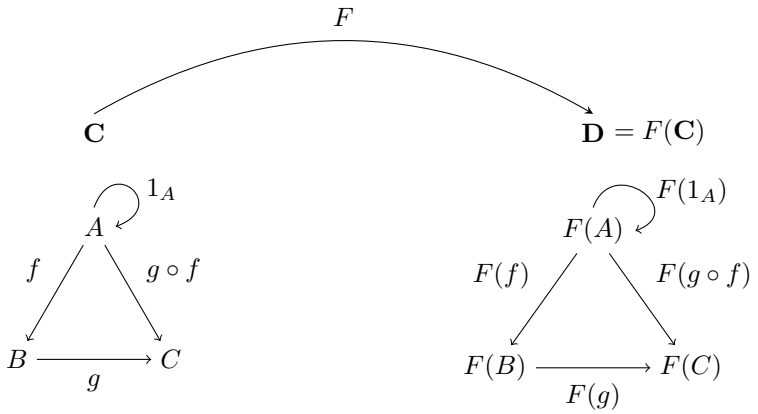


Figure 1.1: Diagram of functor F on categories $\mathbf{C}$ and $\mathbf{D}$.

Since functors are themselves mappings, we can explore the concepts of functors being "1-1" and "onto", called faithful and full respectively. First, some notation. Given a category, $\mathbf{C}$, and objects A, B in that category, we let $\mathrm{Hom}_{\mathbf{C}}(A, B)$ represent the set of all arrows in $\mathbf{C}$ from A to B. Given a functor F from category $\mathbf{C}$ to category $\mathbf{D}$ we define the map $F_{A,B}$ as follows.

$$F_{A,B}\colon\ \mathrm{Hom}_{\mathbf{C}}(A, B) \longrightarrow \mathrm{Hom}_{\mathbf{D}}\big(F(A), F(B)\big)$$
$$f \longmapsto F(f)$$

The functor F is then said to be **full** if $F_{A,B}$ is onto for all objects $A, B \in \mathbf{C}$, and F is said to be **faithful** if $F_{A,B}$ is 1-1 for all objects $A, B \in \mathbf{C}$. Just like a function being both 1-1 and onto yields nice properties, so too does a functor being both full and faithful. Specifically, these are key concepts in the notion of two categories being equivalent. Though there are several definitions of what it means for two categories to be equivalent, we take the following as our definition. Categories $\mathbf{C}$ and $\mathbf{D}$ are **equivalent** if there is a functor $F : \mathbf{C} \to \mathbf{D}$ such that F is full, faithful, and for every object $D \in \mathbf{D}$ there is an object $C \in \mathbf{C}$ such that $F(C)$ is isomorphic to D. This latter property is sometimes referred to as F being "essentially onto", or that categories $\mathbf{C}$ and $\mathbf{D}$ have functors which are "almost inverses" of each other. It is thus called because given any object in $\mathbf{D}$, there is always a corresponding object in $\mathbf{C}$ to which we can apply our functor and get back an object "essentially" the same as our original – one isomorphic to it.

Intuitively, equivalence of categories is the key concept in exploring similarities between two categories. As one might expect to be the case based on the name, all category-theoretic properties are preserved under equivalence of categories. For instance, initial objects, terminal objects, and limits to name a few. These are beyond the scope of this dissertation, however, so we do not elaborate on them here. What we do rely upon is the fact that once we have equivalence of categories, many of the associated properties of the given structures we are examining will transfer over quite nicely.

Chapter 2

Computable Categoricity of Partial Injection Structures

In this chapter, we wish to examine the computable categoricity of partial injection structures. In [7] Cenzer, Harizanov, and Remmel studied computability-theoretic properties of injection structures and their isomorphisms. In [6] Calvert, Cenzer, Harizanov, and Morozov researched computable categoricity of equivalence structures. We build off the work of both by defining partial injection structures, examining what the structures look like, and exploring the properties of computable isomorphisms between them.

2.1 Injection Structures and Partial Injection Structures

An **injection structure** $\mathcal{A} = (A, f)$ consists of a set A and an injection (a 1-1 function) $f : A \to A$. We call an injection structure $\mathcal{A} = (A, f)$ a **computable injection structure** if A is a computable set and f is a total computable function. In [7] Cenzer, Harizanov, and Remmel studied injection structures and their isomorphisms extensively. Since the injection, f, is the sole distinguishing characteristic in these structures, we wish to examine what happens to these partial computable injection structures as we apply f to the elements of our set A. Given an injection structure $\mathcal{A} = (A, f)$, with $a \in A$, we follow the notation in [7] and define the **orbit of** a **under** f to be:

$$\mathcal{O}_f(a) = \{b \in A : \exists n \in \mathbb{N}[f^n(a) = b \vee f^n(b) = a]\}$$

Note that since we wish to follow the usual conventions and restrict our universe to the natural numbers, we have symmetrized the above definition of an orbit accordingly.

Orbits may be finite or infinite in size. And furthermore, injection structures can be completely classified up to isomorphism by the sizes and types of their orbits. From [7] we have the following results on the computable categoricity of injection structures.

Theorem 2.1 (Cenzer, Harizanov, Remmel, [7]). *Let $\mathcal{A} = (A, f)$ be a computable injection structure. Then $\mathcal{A}$ is computably categorical iff $\mathcal{A}$ has finitely many infinite orbits.*

Theorem 2.2 (Cenzer, Harizanov, Remmel, [7]). *Let $\mathcal{A} = (A, f)$ be a computable injection structure. Then $\mathcal{A}$ is computably categorical iff $\mathcal{A}$ is relatively computably categorical.*

In [7] Cenzer, Harizanov, and Remmel also extended the notion of injection structures to examine Σ^0_1 and Π^0_1 injection structures, $\mathcal{A} = (A, f)$, so defined when the universe A is a Σ^0_1 or Π^0_1 set respectively. In these examinations the injection remained a total function. We now take this idea further, and examine what happens when we leave the universe, A as-is, but change the injection f to be only a partial function.

We define here a **partial injection structure** $\mathcal{A} = (A, f)$ to be a structure which consist of a set $A \subseteq \mathbb{N}$ and a partial 1-1 function $f : A \to A$. We think of a **partial 1-1 function**, or a **partial injection**, as a function, f, where for any $x, y \in \mathbb{N}$ if $f(x) \downarrow = f(y)$ then $x = y$, but it is not necessarily true that $\mathrm{dom}(f) = A$. For our purposes, we wish to focus on the case when A is infinite. We call a partial injection structure $\mathcal{A} = (A, f)$ a **partial computable injection structure** if A is a computable set and f is a partial computable function.

We now examine computability-theoretic properties of partial injection structures and their isomorphisms. We note first a slight abuse of notation in the following sections. We will call a partial computable injection structure $\mathcal{A}$ computably categorical iff every partial computable injection structure isomorphic to $\mathcal{A}$ is computably isomorphic to $\mathcal{A}$. Traditionally, the definition of computably categorical requires that the structures themselves be computable, not just partial computable. However, we can use the techniques of Marker's extensions ([33]) to yield a partial injection structure which is indeed computable, and to which all the properties of the structure transfer as expected.

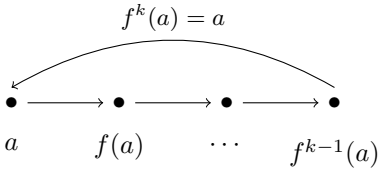
2.2 Classifying the Orbits

We first begin by exploring what happens to these partial computable injection structures, $\mathcal{A} = (A, f)$, as we apply f to the elements of our set A. Given a partial injection structure $\mathcal{A} = (A, f)$,

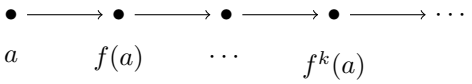
with $a \in A$, we can define the orbit of an element a under f as we did above for injection structures.

This yields the following five different types of orbits, described below. Note that in our descriptions we use ω to denote the natural numbers with their usual ordering, and we use ω^* to denote the reverse ordering of ω. Additionally, we use standard notation where S represents the successor function.

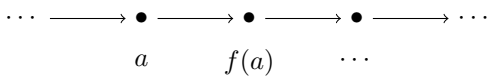
- **Cycles**: These orbits are of the form $\mathcal{O}_f(a) = \{a, f(a), \ldots, f^{k-1}(a)\}$ for some a such that $f^k(a) = a$. Note that in this case we have that both $a \in \text{dom}(f)$ and $a \in \text{range}(f)$. These orbits "cycle back" on themselves at some finite point, and hence are of some finite size, k. They look as follows.



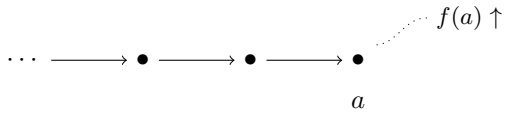
- **ω-orbits**: These orbits go infinitely forward and hence are isomorphic to (ω, S). These orbits are of the form $\mathcal{O}_f(a) = \{f^n(a) : n \in \mathbb{N}\}$ for some $a \notin \text{range}(f)$. The orbits looks as follows.



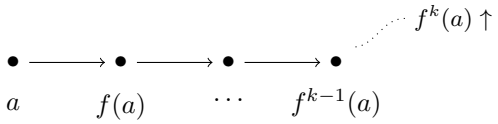
- **Z-orbits**: These orbits go infinitely forward and infinitely backwards, and are therefore isomorphic to $(\mathbb{Z}, S)$. Every element of the orbit is in both $\text{range}(f)$ and $\text{dom}(f)$. Each Z-orbit looks as follows.



- **ω^*-orbits**: These orbits go infinitely backwards and hence are isomorphic to (ω^*, S). The orbits are of the form $\mathcal{O}_f(a) = \{b : \exists n(f^n(b) = a)\}$ for some $a \notin \text{dom}(f)$, that is for some a such that $f(a)\uparrow$. These orbits look as follows.



- **Finite Chains**: These orbits neither go infinitely forward, nor infinitely backward, nor do they cycle back on themselves. These orbits are of some finite size, k, and are of the form $\mathcal{O}_f(a) = \{a, f(a), \ldots, f^{k-1}(a)\}$ for some $a \notin \text{range}(f)$ such that $f^k(a) \notin \text{dom}(f)$, that is such that $f^k(a)\uparrow$. Each finite chain orbit looks as follows.



We note first that the ω^*-orbits and the finite chain orbits exist because in partial injection structures, there may be elements in our universe A which are not in the domain of our function. We note that this is not true for (total) injection structures, where the injection f is a total function. Therefore, total injection structures can only have cycle orbits, ω-orbits, or Z-orbits. We also note that ω-orbits, Z-orbits, and ω^*-orbits are the only orbits of infinite size, and that cycles and finite chains are the only orbits of finite size. We will often call a finite cycle with k elements a **k-cycle**, and a finite chain orbit with k elements a **k-chain**, to emphasize the sizes of the orbits.

Partial injection structures are distinguished from each other solely by their partial injection, f. Partial injection structures can therefore be classified up to isomorphism by describing the number of infinite orbits of each type (ω-orbit, Z-orbit, or ω^*-orbit), and the number of finite orbits of size k for each type (cycles or finite chains) for each finite k. Formally, we define an **isomorphism between partial injection structures** to be a 1-1 and onto function which preserves the injection. By this we mean that if $\mathcal{A} = (A, f)$ and $\mathcal{B} = (B, g)$ are two partial injection structures, they are isomorphic iff there exists a 1-1 and onto fucntion $h : A \to B$ such that $h(f(a)) = g(h(a))$. Or in other words, the following diagram commutes.

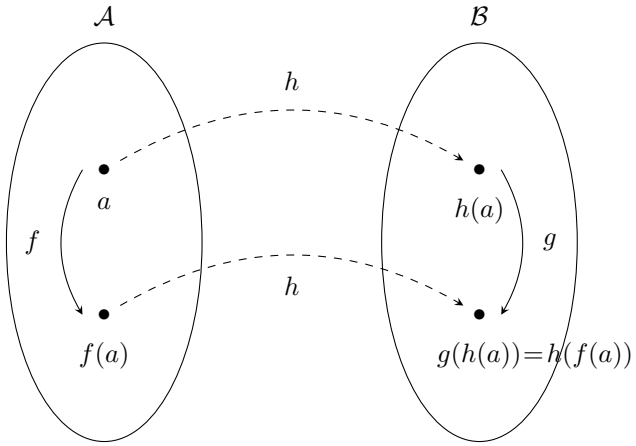


Figure 2.1: Diagram of isomorphic partial injection structures, $(A, f) \simeq (B, g)$ via h.

We now take the opportunity to examine the complexity of the various orbits and other natural properties of partial computable injection structures.

Lemma 2.3. *Let $\mathcal{A} = (A, f)$ be a partial computable injection structure. Then we have the following*

complexities in the arithmetical hierarchy:

1. *dom*(f) *is a* Σ_1^0 *set*
2. *range*(f) *is a* Σ_1^0 *set*
3. $\mathcal{O}_f(a)$ *is a* Σ_1^0 *set*
4. $\{a : \mathcal{O}_f(a)$ *is infinite*$\}$ *is a* Π_2^0 *set.*
5. $\{a : \mathcal{O}_f(a)$ *has type* $Z\}$ *is a* Π_2^0 *set.*
6. $\{a : \mathcal{O}_f(a)$ *has type* $\omega\}$ *is a* D_2^0 *set.*
7. $\{a : \mathcal{O}_f(a)$ *has type* $\omega^*\}$ *is a* D_2^0 *set.*
8. $\{a : \mathcal{O}_f(a)$ *is a cycle*$\}$ *is a* Σ_1^0 *set.*
9. $\{a : \mathcal{O}_f(a)$ *is a finite chain*$\}$ *is a* Σ_2^0 *set.*
10. $\{(k, a) : card\big(\mathcal{O}_f(a)\big) \geq k\}$ *is a* Σ_1^0 *set.*

Proof. The proof here is straightforward. We know that A is a computable set and f is a partial computable function. Furthermore, we know that given a time bound s on f, f_s is therefore a total computable function. To prove each of the above statements, we note the following characterizations.

1. Domain of f:

$$a \in \text{dom}(f) \iff f(a)\downarrow \iff \exists t f_t(a)\downarrow$$

   Therefore dom(f) is a Σ_1^0 set.

2. Range of f:

$$a \in \text{range}(f) \iff \exists b \exists t f_t(b) = a$$

   Therefore, range(f) is a Σ_1^0 set.

3. Orbits:

$$\mathcal{O}_f(a) = \{b : \underbrace{\exists n \exists t \big[f_t^n(b) = a \vee f_t^n(a) = b\big]}_{\Sigma_1^0}\}$$

4. Infinite orbits:

$$\begin{aligned} a \in \text{infinite orbit} &\iff a \text{ goes infinitely forwards (and not a cycle)} \\ &\quad \vee\ a \text{ goes infinitely backwards (and not a cycle)} \\ &\iff \big[\forall m \exists s\ f_s^m(a)\downarrow \neq a\big]\ \vee\ \big[\forall k \exists b \exists t\ f_t^k(b) = a \wedge b \neq a\big] \\ &\iff \forall m \forall k \exists b \exists t \exists s \big[f_s^m(a)\downarrow \neq a\ \vee\ \big(f_t^k(b) = a \wedge b \neq a\big)\big] \end{aligned}$$

This is a Π^0_2 statement, hence $\{a : \mathcal{O}_f(a) \text{ is infinite}\}$ is a Π^0_2 set.

5. Z-orbits:

$$\begin{aligned} a \in Z\text{-orbit} &\iff a \text{ goes infinitely forwards}\ \wedge\ a \text{ goes infinitely backwards} \\ &\iff \big[\forall m \exists s\ f_s^m(a)\downarrow \neq a\big]\ \wedge\ \big[\forall k \exists b \exists t\ f_t^k(b) = a\big] \\ &\iff \forall m \forall k \exists b \exists t \exists s \big[f_s^m(a)\downarrow \neq a\ \wedge\ f_t^k(b) = a\big] \end{aligned}$$

This is a Π^0_2 statement, hence $\{a : \mathcal{O}_f(a) \text{ has type } Z\}$ is a Π^0_2 set.

6. ω-orbits:

$$\begin{aligned} a \in \omega\text{-orbit} &\iff a \text{ goes infinitely forwards}\ \wedge\ a \text{ does } \underline{\text{not}} \text{ go infinitely backwards} \\ &\iff \underbrace{\big[\forall k \exists t\ f_t^k(a)\downarrow \neq a\big]}_{\Pi^0_2}\ \wedge\ \underbrace{\big[\exists k \forall b \forall t\ f_t^k(b) \neq a\big]}_{\Sigma^0_2} \end{aligned}$$

Therefore $\{a : \mathcal{O}_f(a) \text{ has type } \omega\} = \Pi^0_2\text{-set} \cap \Sigma^0_2\text{-set}$, and hence $\{a : \mathcal{O}_f(a) \text{ has type } \omega\}$ is D^0_2.

7. ω^*-orbits:

$$\begin{aligned} a \in \omega^*\text{-orbit} &\iff a \text{ does } \underline{\text{not}} \text{ go infinitely forwards}\ \wedge\ a \text{ goes infinitely backwards} \\ &\iff \underbrace{\big[\exists k \forall t\ f_t^k(a)\uparrow\big]}_{\Sigma^0_2}\ \wedge\ \underbrace{\big[\forall k \exists b \exists t\ f_t^k(b) = a\big]}_{\Pi^0_2} \end{aligned}$$

Therefore $\{a : \mathcal{O}_f(a) \text{ has type } \omega^*\} = \Sigma^0_2\text{-set} \cap \Pi^0_2\text{-set}$, and hence $\{a : \mathcal{O}_f(a) \text{ has type } \omega^*\}$ is D^0_2.

8. Cycles:

$$\begin{aligned} a \in \text{cycle} &\iff \text{at some point } a \text{ cycles back on itself} \\ &\iff \exists k \exists t f_t^k(a) = a \end{aligned}$$

This is a Σ_1^0 statement, hence $\{a : \mathcal{O}_f(a) \text{ is a cycle }\}$ is a Σ_1^0 set.

9. Finite Chains:

$$\begin{aligned} a \in \text{finite chain} &\iff a \text{ does } \underline{\text{not}} \text{ go infinitely forwards } \wedge\ a \text{ does } \underline{\text{not}} \text{ go infinitely backwards} \\ &\iff \big[\exists m \forall s\ f_s^m(a) \uparrow\big]\ \wedge\ \big[\exists k \forall b \forall t\ f_t^k(b) \neq a\big] \\ &\iff \exists m \exists k \forall b \forall t \forall s \big[f_s^m(a) \uparrow\ \wedge\ f_t^k(b) \neq a\big] \end{aligned}$$

This is a Σ_2^0 statement, hence $\{a : \mathcal{O}_f(a) \text{ is a finite chain }\}$ is a Σ_2^0 set.

10. Minimum Cardinality:

$$\begin{aligned} \text{card}\big(\mathcal{O}_f(a)\big) \geq k &\iff \exists \text{chain of length } k \text{ which contains } a \\ &\iff \exists b \exists t_1 \ldots \exists t_k \exists s \Big[\big(f_{t_1}(b) \downarrow \neq b\ \wedge\ f_{t_2}^2(b) \downarrow \neq b\ \wedge\ \ldots\ \wedge\ f_{t_k}^k(b) \downarrow \neq b\big) \wedge \\ &\qquad \big(b = a\ \vee\ f_s(b) = a\ \vee\ \ldots\ \vee\ f_s^k(b) = a\big)\Big] \end{aligned}$$

This is a Σ_1^0 statement, hence $\{(k, a) : \text{card}\big(\mathcal{O}_f(a)\big) \geq k\}$ is a Σ_1^0 set.

□

2.3 Relatively Computably Categorical Partial Injection Structures

We now examine relative computable categoricity of partial injection structures. We present two different theorems, each establishing relative computable categoricity for partial computable injection structures with certain types of orbits.

We first recall Theorem 2.1 and Theorem 2.2 from Cenzer, Harizanov, Remmel in [7]. Together, these theorems give that any computable injection structure with finitely many infinite orbits is relatively computably categorical. For injection structures, the infinite orbits can be only Z-orbits

or ω-orbits. We now extend this theorem for partial injection structures. Partial injection structures have an additional type of infinite orbit, namely ω^*-orbits, which we will take into account here. Additionally, partial injection structures also have finite chain orbits, which injection structures do not. Our corresponding theorem for partial injection structures also requires finitely many of the finite chain orbits.

Theorem 2.4. *If $\mathcal{A} = (A, f)$ is a partial computable injection structure with finitely many infinite orbits and finitely many finite chain orbits, then $\mathcal{A}$ is relatively computably categorical.*

Proof. Note that the proof of this follows closely that of the forward direction of Theorem 2.1, given in [7]. Let $\mathcal{A} = (A, f)$ be a partial computable injection structures with finitely many infinite orbits and finitely many finite chain orbits as follows:

- m ω-orbits with representatives $a_1, \ldots, a_m$ s.t. $a_i \notin \text{range}(f)$
- n ω^*-orbits with representatives $b_1, \ldots, b_n$ s.t. $b_i \notin \text{dom}(f)$
- l Z-orbits with representatives $z_1, \ldots, z_l$ s.t. $\mathcal{O}_f(z_i) \neq \mathcal{O}_f(z_j)$ for $i \neq j$
- p finite chain-orbits with representatives $c_1, \ldots, c_p$ s.t. $c_i \notin \text{dom}(f)$

In [22] Goncharov proved that $\mathcal{A}$ is relatively computably categorical iff $\mathcal{A}$ has a formally c.e. Scott family of formulas. Therefore, to proceed, we will come up with a formally c.e. Scott family for $\mathcal{A}$.

Let $\vec{d} = d_0, \ldots, d_r$ be a sequence in A distinct from $a_1, \ldots, a_m$, from $b_1, \ldots b_n$, from $z_1, \ldots, z_l$, and from $c_1, \ldots, c_p$. Then $\forall q \leq r$ we have that either d_q is part of a cycle, a finite chain, an ω-orbit, an ω^*-orbit, or a Z-orbit. This corresponds to one of the following:

(1) $\exists t\, f_t^k(d_q) = d_q$ (for some fixed, minimal $k > 0$)

(2) $\exists t\, f_t^k(d_q) \downarrow = c_i$ (for some fixed $k \geq 0, i \in \{1, \ldots, p\}$)

(3) $\exists t\, f_t^k(a_i) = d_q$ (for some fixed $k \geq 0, i \in \{1, \ldots, m\}$)

(4) $\exists t\, f_t^k(d_q) = b_i$ (for some fixed $k \geq 0, i \in \{1, \ldots, n\}$)

(5) $\exists t\, f_t^k(d_q) = z_i$ (for some fixed $k \geq 0, i \in \{1, \ldots, l\}$)

(6) $\exists t\, f_t^k(z_i) = d_q$ (for some fixed $k > 0, i \in \{1, \ldots, l\}$)

Note that in (1) if we have more than one element in our sequence $\vec{d}$ which meets this condition for the same value of k, then we have to specify whether they are in the same cycle or different cycles. Therefore for d_q, d_Q in our sequence $\vec{d}$ with $q \neq Q$ and $q, Q \leq r$, if both d_q and d_Q satisfy the same formula (1) with the same value of k then either:

(7) $\exists t\, f_t^j(d_q) \downarrow = d_Q$ (for some fixed $j < k$)

(8) $\forall j < k\ \exists t\, f_t^j(d_q) \downarrow \neq d_Q$

Note that each of the above statements, (1)-(8), are c.e. since f is partial computable, and in each case we have added a time bound, t. This is obvious for (2)-(7). To see this more clearly for (1) we can translate "k minimal" into simply adding the following to the end of our statement: $\exists s(f_s(d_q) \downarrow \neq d_q \wedge f_s^2(d_q \downarrow \neq d_q) \wedge \cdots \wedge f_s^{k-1}(d_q) \downarrow \neq d_q)$. And for (8) we can translate "$\forall j < k$" into replacing (8) with the following: $\exists t(f_t^0(d_q) \downarrow \neq d_Q \wedge f_t^1(d_q) \downarrow \neq d_Q \wedge \cdots \wedge f_t^{k-1}(d_q) \downarrow \neq d_Q)$.

Therefore, each formula in our Scott family of formulas will look like some combination of (1)-(8) with d_q replaced with the variable x_q, and d_Q replaced with the variable x_Q. It is easy to see that each sequence $\vec{d}$ in A will have a formula in this family – we simply take the appropriate combination of (1)-(8).

Therefore, to complete the proof, we need only show that given two sequences, $\vec{d}$ and $\vec{e}$, in A, if they satisfy the same Scott formula, then there exists an automorphism, h, such that $h(\vec{d}) = \vec{e}$. We let $\vec{d} = d_1, \ldots, d_r$ and $\vec{e} = e_1, \ldots, e_r$ be two such sequences in A. We first define a partial function h as follows:

$$h(x) = \begin{cases} x & \text{if } x \text{ is in an infinite orbit or finite chain} \\ x & \text{if } x \text{ is in a cycle distinct from all } d_1, \ldots, d_r \text{ and } e_1, \ldots, e_r \\ f^j(e_q) & \text{if } x \text{ is in } k\text{-cycle and } x = f^j(d_q) \text{ for some } j < k \end{cases}$$

It is clear then that for x in an infinite orbit (ω-orbit, Z-orbit, or ω^*-orbit), in a finite chain, or in a cycle distinct from all of $d_1, \ldots, d_r$ and $e_1, \ldots, e_r$, that h is the identity function, and therefore is 1-1 and preserves these orbits. For x in one of the cycle orbits with some d_q, it is easy to see that the Scott formulas for $\vec{d}$ and $\vec{e}$ from (1) and from either (7) or (8), as appropriate, will ensure that the orbits are preserved under h, and that h is 1-1. Finally, then, we must extend h from a partial function to a total one to get our desired automorphism. We do so by defining h on inputs from k-cycles with some e_q, but with no d_Q. That is, x for which $x = f^j(e_q)$ for some $j < k$ and $x \neq f^J(d_Q)$ for any $J < k$ and $Q \in \{1, \ldots, r\}$. To finish defining h, then, we simply send each of the e_q's to one of the d_Q's, say, take the smallest q for which $\mathcal{O}_f(e_q)$ has not been mapped under h to the smallest d_Q for which $\mathcal{O}_f(d_Q) \notin \text{range}(h)$. (Note: we know that there are indeed the same number of such e_q's as d_Q's. If there are K-many e_q's in k-cycles without any d_Q's, then so, too must there be K-many d_Q's in cycles without e_q's.) Then define $h(f^j(e_q)) = f^j(d_Q)$. This extension of h is

therefore also 1-1 and preserves orbits (takes k-cycles to k-cycles), and it completes the extension so that h is additionally now total and onto. Hence h as defined is the required automorphism. $\square$

At this point one might conjecture, then, that finite chain orbits in partial injection structures behave much like infinite orbits do in injection structures, as they too have no guaranteed stage at which the "endpoint" of an orbit is reached. In some sense, as with the previous theorem, this is indeed true. However, finite chain orbits are not actually infinite orbits, and so they do indeed have some finite size. In some sense therefore they also behave like finite cycle orbits. We see this with the following theorem.

We note that for injection structures, Theorem 2.1, which is the corresponding theorem to the above, gave the only type of computably categorical injection structures — those with finitely many infinite orbits. The same is not true for partial injection structures, however. The finite chain orbits can also, under certain conditions, behave like finite cycle orbits. We therefore obtain another type of relatively computably categorical partial injection structure: one in which infinitely many finite chain orbits are allowed under certain conditions.

Theorem 2.5. *If $\mathcal{A} = (A, f)$ is a partial computable injection structure made up of only finite orbits (no ω-orbits, no $\omega*$-orbits, and no Z-orbits), and furthermore, $\mathcal{A}$ has infinitely many finite chain orbits of one size (call it ℓ), finitely many finite chain orbits of size other than ℓ, and finitely many cycles of size $\geq \ell$, then $\mathcal{A}$ is relatively computably categorical.*

Proof. This proof follows closely that for Theorem 2.4. We proceed by building the Scott family.

Let $\mathcal{A} = (A, f)$ be a partial computable injection structures with no infinite orbits, infinitely many finite chain orbits of size ℓ, finitely many finite chain orbits of size $\neq \ell$, and finitely many cycle orbits of size $\geq \ell$ as follows:

- 0 ω-orbits
- 0 ω^*-orbits
- 0 Z-orbits
- infinitely many finite chain orbits of size ℓ
- n finite chain-orbits of size $\neq \ell$ with representatives $a_1, \ldots, a_n$ s.t. $a_i \notin \text{range}(f)$
- m finite cycle orbits of size $\geq \ell$ with representatives $b_1, \ldots, b_m$
- any number of finite cycle orbits of size $< \ell$

We again use Goncharov's result in [22] that $\mathcal{A}$ is relatively computably categorical iff $\mathcal{A}$ has a formally c.e. Scott family. Therefore, to proceed, we will come up with a formally c.e. Scott family for A.

Let $\vec{d} = d_0, \ldots, d_r$ be a sequence in A distinct from $a_1, \ldots, a_n$ and from $b_1, \ldots, b_m$. Then $\forall q \leq r$ we have that either d_q is part of a finite chain of size ℓ, finite chain of size other than ℓ, cycle of size $\geq \ell$, or cycle of size $< \ell$. This corresponds to one of the following:

(1) $\exists c_1 \ldots \exists c_\ell \exists t_1 \ldots \exists t_{\ell-1}$ such that all of the following:

- $\forall i \neq j \in \{1, \ldots, \ell\}\ c_i \neq c_j$
- $\forall i \in \{1, \ldots, \ell\} \forall j \in \{1, \ldots, n\}\ \ c_i \neq a_j$
- $\forall i \in \{1, \ldots, \ell\} \forall j \in \{1, \ldots, m\}\ \ c_i \neq b_j$
- $d_q = c_i$ for some $i \in \{1, \ldots, \ell\}$
- $f_{t_1}(c_1) = c_2\ \wedge\ f_{t_2}(c_2) = c_3\ \wedge\ \cdots\ \wedge\ f_{t_{\ell-1}}(c_{\ell-1}) = c_\ell$

(2) $\exists t\, f_t^k(a_i) = d_q$ (for some fixed $k \geq 0, i \in \{1, \ldots, n\}$)

(3) $\exists t\, f_t^k(d_q) = b_i$ (for some fixed minimal $k \geq \ell, i \in \{1, \ldots, m\}$)

(4) $\exists t\, f_t^k(d_q) = d_q$ (for some fixed minimal $k < \ell$)

Note that in (4) if we have more than one element in our sequence $\vec{d}$ which meets this condition for the same value of k, then we have to specify whether they are in the same cycle or different cycles. Therefore for d_q, d_Q in our sequence $\vec{d}$ with $q \neq Q$ and $q, Q \leq r$, if both d_q and d_Q satisfy the same formula (4) with the same value of k then either:

(5) $\exists t\, f_t^j(d_q) \downarrow = d_Q$ (for some fixed $j < k$)

(6) $\forall j < k\ \exists t\, f_t^j(d_q) \downarrow \neq d_Q$

Note that each of the above statements, (1)-(6), are c.e. since f is partial computable and in each case we have added a time bound, t. Additionally, each "$\forall$" in the above formulas is bounded by some fixed number and can therefore be expanded to remove the "$\forall$"s accordingly. We can furthermore translate "k minimal" into simply adding the following to the end of our statement: $\exists s_1 \cdots \exists s_{k-1} \Big[f_{s_1}(d_q) \downarrow \neq (*)\ \wedge\ f_{s_2}^2(d_q) \downarrow \neq (*)\ \wedge \cdots \wedge\ f_{s_{k-1}}^{k-1}(d_q) \downarrow \neq (*) \Big]$, where $(*)$ represents b_i or d_q as appropriate.

Therefore, each formula in our Scott family of formulas will look like some combination of (1)-(6) with d_q replaced with the variable x_q, and d_Q replaced with the variable x_Q. It is easy to see that

each sequence $\vec{d}$ in A will have a formula in this family – we simply take the appropriate combination of (1)-(6).

Therefore, to complete the proof, we need only show that given two sequences, $\vec{d}$ and $\vec{e}$, in A, if they satisfy the same Scott formula, then there exists an automorphism, h, such that $h(\vec{d}) = \vec{e}$. We let $\vec{d} = d_1, \ldots, d_r$ and $\vec{e} = e_1, \ldots, e_r$ be two such sequences in A. We first define a partial function h as follows:

$$h(x) = \begin{cases} x & \text{if } x \text{ is in finite chain orbit size } \neq \ell \\ x & \text{if } x \text{ is in cycle orbit size } \geq \ell \\ x & \text{if } x \text{ is in } \ell\text{-chain distinct from } d_1, \ldots, d_r, e_1, \ldots, e_r \\ f^j(e_q) & \text{if } x \text{ is in } \ell\text{-chain and } x = f^j(d_q) \text{ for some } j < \ell \\ y \text{ s.t. } f^j(y) = e_q & \text{if } x \text{ is in } \ell\text{-chain and } f^j(x) = d_q \text{ for some } j < \ell, j > 0 \\ x & \text{if } x \text{ is in a } k\text{-cycle } (k < \ell) \text{ distinct from } d_1, \ldots, d_r, e_1, \ldots, e_r \\ f^j(e_q) & \text{if } x \text{ is in a } k\text{-cycle } (k < \ell) \text{ and } x = f^j(d_q) \text{ for some } j < k \end{cases}$$

It is clear then that for x in a finite chain orbit of size $\neq \ell$, in a cycle of size $\geq \ell$, in an ℓ-chain distinct from all of $d_1, \ldots, d_r$ and $e_1, \ldots, e_r$, or in a cycle of size $< \ell$ distinct from all of $d_1, \ldots, d_r$ and $e_1, \ldots, e_r$, that h is the identity function, and therefore is 1-1 and preserves these orbits. For x in one of the ℓ-chains or in a cycle of size $< \ell$ that contains one of $d_1, \ldots, d_r$ it is easy to see that the Scott formulas for $\vec{d}$ and $\vec{e}$ from (1), (4) and from either (5) or (6), as appropriate, will ensure that the orbits are preserved under h, and that h is 1-1. Finally, then, we must extend h from a partial function to a total one to get our desired automorphism. We do so by defining h on inputs in ℓ-chains and "$<\ell$"-cycles with some e_q, but with no d_Q. For x in such "$<\ell$"-cycles with $x = f^j(e_q)$ for some $j < k < \ell$, we extend h in a similar way as we did in Theorem 2.4 — that is, $h(f^j(e_q)) = f^j(d_Q)$ for the next smallest q and Q with d_Q not yet in range(h). For x in ℓ-chains, we extend h by again sending the entire chain containing e_q to the entire chain containing d_Q (the chain with the smallest q for e_q not yet mapped under h to the chain with the smallest Q for d_Q not yet in range(h)), ensuring that we map the beginning of e_q's ℓ-chain to the beginning of d_Q's ℓ-chain, regardless of whether e_q and d_Q are the same distance from the beginning of their respective chains. This extension of h is therefore 1-1 and also preserves orbits (takes k-cycles to k-cycles and ℓ-chains to ℓ-chains, ensuring the respective parts of each chain are mapped accordingly). This completes the extension of h, so it is now additionally total and onto, hence h defines the required automorphism.

□

Since we know that relative computable categoricity implies computable categoricity, the following corollary is immediate.

Corollary 2.6. *If $\mathcal{A} = (\mathbb{N}, f)$ is a partial computable injection structure satisfying either of the following, then $\mathcal{A}$ is computably categorical.*

1. *$\mathcal{A}$ has finitely many infinite orbits and finitely many finite chain orbit; or*
2. *$\mathcal{A}$ has no infinite orbits, infinitely many finite chain orbits of one size (call it ℓ), finitely many finite chain orbits of size other than ℓ, and finitely many cycles of size $\geq \ell$.*

We now have two classes of partial injection structures which are computably categorical. The natural question, of course, is then: are these the only kinds of computably categorical partial injection structures? We can continue on to examine other types of partial injection structures and whether or not they are computably categorical. We do so in the following section.

2.4 Non-Computably Categorical Partial Injection Structures

We now wish to examine types of partial injection structures which are not computably categorical. Specifically, we wish to examine what happens for partial injection structures which are not the ones as in Theorem 2.4 and Theorem 2.5. We will construct counter-examples for most aspects of the other cases.

Firstly, of course, we must determine what the other cases are. We have given conditions for relative computable categoricity, and hence computable categoricity for the following types of partial injection structures, $\mathcal{A} = (A, f)$:

- $\mathcal{A}$ has finitely many infinite orbits and finitely many finite chain orbit
- $\mathcal{A}$ has no infinite orbits, infinitely many finite chain orbits of one size (call it ℓ), finitely many finite chain orbits of size other than ℓ, and finitely many cycles of size $\geq \ell$.

The other cases which we have not yet discussed are then:

- Infinitely many infinite orbits
- Infinitely many finite chain orbits such that:
  - There are infinitely many finite chain orbits of only one size, ℓ, and ...

- any number (finite or infinite) of infinite orbits, or
- infinitely many cycles of size $\geq \ell$.

– There are infinitely many finite chain orbits of two or more sizes.

– There are infinitely many finite chains of arbitrarily large size.

Note that in the final two categories above, we mean the following. Each of the two sizes of finite chains has infinitely many finite chains in it — for instance, infinitely many ℓ-chains and infinitely many m-chains, for some $\ell, m < \omega$. The infinitely many finite chains of arbitrarily large size may have possibly only finitely many of each finite size, but must have arbitrarily many such sizes – for instance: one 1-chain, and one 2-chain, and one 3-chain, and so on.

We now proceed here to provide counter examples of types of partial injection structures within each of the preceding categories. We begin first by examining partial injection structures with infinitely many infinite orbits. Cenzer, Harizanov, and Remmel proved that any computable injection structure with infinitely many infinite orbits is not computably categorical. We get a similar result for partial computable injection structures if we restrict ourselves to partial injection structures with only infinitely many infinite orbits of one type and nothing else.

Theorem 2.7. *If $\mathcal{A}$ is a partial computable injection structure with infinitely many infinite orbits of one type (Z-, ω-, or ω^*-), and no other types of orbits, then $\mathcal{A}$ is not computably categorical.*

Proof. In [7] Cenzer, Harizanov, and Remmel constructed computable injection structures with infinitely many Z-orbits which were not computably categorical, and they also constructed computable injection structures with infinitely many ω-orbits which were not computably categorical. Since a computable injection structure is also a partial computable injection structure, these same constructions will build partial computable injection structures with infinitely many Z-orbits, or with infinitely many ω-orbits, which are also not computably categorical. Therefore, to complete the proof for partial computable injection structures, we need only examine the case where our partial computable injection structure has infinitely many ω^*-orbits.

If we examine in detail the constructions for infinitely many Z-orbits and for infinitely many ω-orbits in [7], we see that the constructions are for the most part identical. Both constructions build a computable injection structure up in stages, each checking the same conditions and building the necessary orbits accordingly. To build infinitely many ω-orbits, we attach elements to the end or right-hand side of the previously built orbits and grow the orbit in the "forward" direction. To build infinitely many Z-orbits, we attach elements to alternating ends of the previously built orbits and grow the orbit in both the "forward" and the "backward" directions.

A straightforward adaptation of this same construction yields the infinitely many ω^*-orbits necessary. We build a partial computable injection structure up in stages, checking the same conditions as before and building the necessary orbits accordingly. In this case, we attach elements to the beginning or left-hand side of the previously built orbits and grow the orbit in the "backward" direction. The details follow as expected, so that in the end we have a partial computable injection structure with infinitely many ω^*-orbits which is not computably categorical.

□

We now examine what happens for partial computable injection structures with infinitely many finite chain orbits of two sizes and no other types of orbits. By this we mean that there are infinitely many k-chains and infinitely many m-chains for some $k < m < \omega$, and there are no other orbits in the structure. Note that if we are following along in the order that we listed all the different possibilities of types of partial computable injection structures, we have skipped a few scenarios. We present the results in the order here, since the various scenarios build upon each other.

Theorem 2.8. *If $\mathcal{A}$ is a partial computable injection structure with infinitely many finite chain orbits of two different sizes and no other types of orbits, then $\mathcal{A}$ is not computably categorical.*

Before writing a formal proof of Theorem 2.8, we proceed first with an example. This example will give the ideas of the methods used in the more general proof, but in a more concrete (and hopefully therefore more understandable) setting.

Example 2.9 (Specific case of Theorem 2.8)**.** Let $\mathcal{A}$ be a partial injection structure with infinitely many finite chain orbits of size 2 and infinitely many finite chain orbits of size 3, and no other orbits of any other types. Then $\mathcal{A}$ is not computably categorical.

Proof of Example 2.9. We assume that $\mathcal{A} = (\mathbb{N}, f)$ with f defined by using multiples of 5, so that "0" and "1" form the 2-chains, and "2","3", and "4" form the 3-chains. Formally we define f as follows for each $n \in \mathbb{N}$: $f(5n) = 5n+1$; $f(5n+1) = \uparrow$; $f(5n+2) = 5n+3$; $f(5n+3) = 5n+4$; $f(5n+4) = \uparrow$. A picture of what f's orbits look like is contained in Figure 2.2.

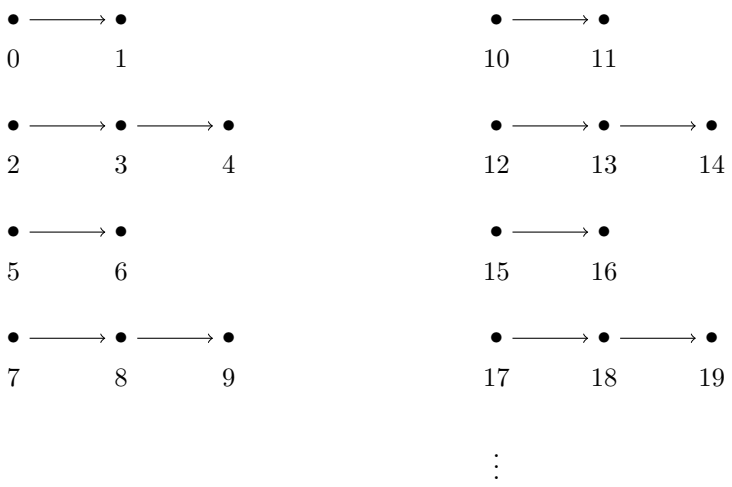


Figure 2.2: Partial computable injection structure $\mathcal{A}$ with infinitely many 2-chains and infinitely many 3-chains in which $\text{dom}(f)$ is computable.

In this case, we have that $\text{dom}(f) = \mathbb{N} - (\{5n+1 : n \in \mathbb{N}\} \cup \{5n+4 : n \in \mathbb{N}\})$, which is a computable set. Our goal will be to construct a partial injection structure, $\mathcal{B} = (\mathbb{N}, g)$, isomorphic to $\mathcal{A}$ in which $\text{dom}(g)$ is not computable.

Let C be some noncomputable c.e. set which does not contain 0. By using the usual conventions (see for instance [42] or [14]), we can assume that there exists a computable sequence of finite sets, $\{C_s\}_{s\geq 0}$, such that $C = \cup_{s\geq 0} C_s$, and such that for all s, $C_s \subseteq \{0, 1, \ldots, s-1\}$, $C_s \subseteq C_{s+1}$, and $\text{card}(C_{s+1} - C_s) \leq 1$.

We will build g up in stages, g_s, so that in the end $g = \lim_s g_s$.

stage 0: We take g_0 to be the function that is completely undefined. That is $\forall x$, let $g_0(x) \stackrel{\text{def}}{=} \uparrow$.

stage s: Assume at the end of stage s we have the following:

- $\forall i \in \{1, \ldots, s\}$ all of the following hold:

  (i) $g_s\big(5(i-1)\big) = 5(i-1)+1$

  (ii) $g_s\big(5(i-1)+4\big) = \uparrow$

  (iii) $g_s\big(5(i-1)+2\big) = 5(i-1)+3$

- $\forall i \in C_s \subseteq \{1, \ldots, s-1\}$ both of the following hold:

  (iv) $g_s\big(5(i-1)+1\big) = 5(s-1)+4$

  (v) $g_s\big(5(i-1)+3\big) = \uparrow$

- and, $\forall i < s$:

  (vi) $5(i-1)+4 \in \text{range}(g_s)$

stage s+1: We assume that at the end of stage s, (i)-(vi) held as appropriate and we now extend g_s to g_{s+1}. Note, that at a previous stage $t < s+1$ we may have defined $g_t(x) = \uparrow$, and at this stage we may define $g_{s+1}(x) \downarrow$. This is ok, as we are building g to be a partial computable function, which may not halt on some inputs until we get to a certain stage.

- First extend g_s to g_{s+1} by defining g_{s+1} on the following inputs:

  $g_{s+1}(5s) \stackrel{\text{def}}{=} 5s+1$

  $g_{s+1}(5s+1) \stackrel{\text{def}}{=} \uparrow$

  $g_{s+1}(5s+2) \stackrel{\text{def}}{=} 5s+3$

- Now, check whether $C_{s+1} - C_s = \emptyset$.

- If $C_{s+1} - C_s \neq \emptyset$ we note that by assumption $\text{card}(C_{s+1} - C_s) \leq 1$, therefore there can be only one element in $C_{s+1} - C_s$, call it $i \in C_{s+1} - C_s$. Define g_{s+1} on the following inputs:

  $g_{s+1}\big(5(i-1)+1\big) \stackrel{\text{def}}{=} 5(s)+4$

  $g_{s+1}(5s+3) \stackrel{\text{def}}{=} \uparrow$

  This has the effect of turning the 2-chain corresponding to $n = (i-1)$ into the 3-chain corresponding to $n = s$, and hence the first part of the $n = s$ 3-chain becomes a 2-chain.

- If $C_{s+1} - C_s = \emptyset$, further extend g_s to g_{s+1} by defining:

  $g_{s+1}(5s+3) \stackrel{\text{def}}{=} 5s+4$

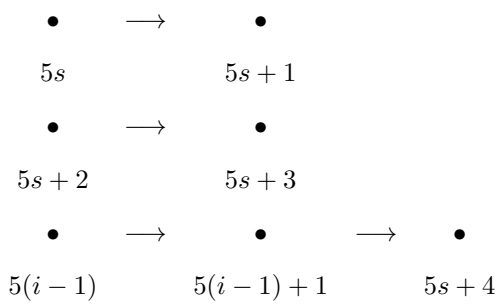
Let $g = \lim_s g_s$.

This completes the construction. To see more intuitively what's happening during the construction, we examine the following pictures.

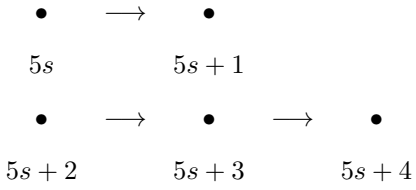
At the beginning of each stage $s+1$ we build the following finite chains. (We have not yet determined whether $5s+3$ will go anywhere):

$$\begin{array}{ccccccc} \bullet & \longrightarrow & \bullet & & & & \\ 5s & & 5s+1 & & & & \\ \bullet & \longrightarrow & \bullet & \longrightarrow & ? & & \\ 5s+2 & & 5s+3 & & & & \end{array}$$

If there is some $i \in C_{s+1} - C_s$ then we complete these finite chains by taking the (previously defined) 2-chain corresponding to $n = i - 1$ and turning it into a 3-chain. We leave the remaining two finite chains as already built:



If there is no $i \in C_{s+1} - C_s$ then we simply complete the already-built finite chains as follows:



We have built g over $\mathbb{N}$, since at each stage, $s+1$, of the construction, we designate what happens to another multiple of natural numbers modulo 5, namely: $5s, 5s+1, 5s+2, 5s+3, 5s+4$. The function g as constructed is indeed well-defined. We note that during the construction, we may initially designate $g_t(x) =\uparrow$ and then at a later stage, $s > t$, designate $g_s(x) \downarrow= y$. This is ok, as we define $g = \lim_s g_s$, and we are building g to be a partial computable function. A partial computable function may not halt for a given input, x, at an initial stage of computation. The key ingredient to the construction is that once $g_s(x) \downarrow= y$, then this remains the case for all later stages.

We see that g is indeed partial computable because we have defined a partial computable process to compute g on input x. Specifically: given input x we follow the above construction; if at some stage $g_s(x) \downarrow= y$, then halt and output $g(x) = y$; otherwise, compute forever. It is also clear that g is a 1-1 function since at each stage of the construction, we never specify $g_s(x) = y$ for two different x's.

Therefore, we let $\mathcal{B} = (\mathbb{N}, g)$, and we have that $\mathcal{B}$ is a partial computable injection structure. Now, it is clear that $\mathcal{A} = (\mathbb{N}, f) \simeq \mathcal{B} = (\mathbb{N}, g)$, since we have $A = B = \mathbb{N}$, and we have built g to have exactly the same numbers, sizes, and types of orbits (namely infinitely many 2-chains, infinitely many 3-chains, and no other types of orbits) as f. To complete the proof then, we need to show

that $\mathrm{dom}(g)$ is not computable. Notice that as constructed,

$$\begin{aligned} x \in \mathrm{dom}(g) \iff & x = 5n+2 \text{ for some } n \in \mathbb{N}, \quad \text{or,} \\ & x = 5n \text{ for some } n \in \mathbb{N}, \quad \text{or,} \\ & x = 5(i-1)+1 \text{ for some } i \in C, \quad \text{or,} \\ & x = 5s+3 \text{ for some } s \text{ such that } C_{s+1} - C_s = \emptyset \end{aligned}$$

Therefore $\mathrm{dom}(g) = \{5n+2 : n \in \mathbb{N}\} \cup \{5n : n \in \mathbb{N}\} \cup \{5(i-1)+1 : i \in C\} \cup \{5s+3 : C_{s+1} - C_s = \emptyset\}$. This is essentially the computable join of four sets: $\mathbb{N}, \mathbb{N}, C$, and $\{s : C_{s+1} - C_s = \emptyset\}$. It is therefore only computable iff all of the sets are computable, but we know that C is not computable. Therefore, $\mathrm{dom}(g)$ is not computable.

Thefore, we have built a computable copy of $\mathcal{A}$ in which the domain of the injection is not computable. Therefore $\mathcal{A}$ is not computably categorical.

Finally, to demonstrate this example even more concretely in Figure 2.3 we give a diagram of what this process would look like using some noncomputable, c.e. set C with $C_0 = C_1 = C_2 = C_3 = C_4 = \emptyset$, $C_5 = \{3\}$, $C_6 = \{3, 5\} = C_7$. We give the explicit construction up through stage 7. The finite chains in red were created and switched during stage 5, the finite chains in blue were created and switched during stage 6.

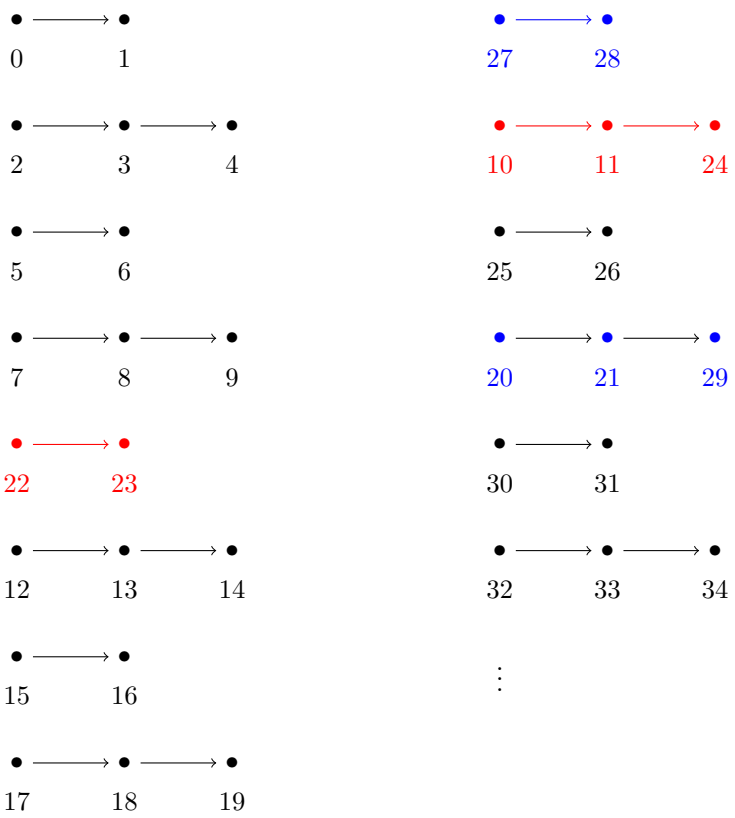


Figure 2.3: Construction through stage 7 of a partial computable injection structure with infinitely many 2-chains and infinitely many 3-chains in which $\mathrm{dom}(g)$ is not computable.

□

We can now proceed with a formal proof of Theorem 2.8, that for finite m and n, a partial computable injection structure with infinitely many m-chains and infinitely many k-chains and no other types of orbits is not computably categorical.

Proof of Theorem 2.8. We let $\mathcal{A}$ be some partial computable injection structure with infinitely many k-chains and infinitely many m-chains for some $k < m < \omega$, and with no other types of orbits. The proof of this follows closely to the proof contained in Example 2.9, with the following modifications. Without loss of generality we assume $\mathcal{A} = (\mathbb{N}, f)$ with f defined as follows so that $\mathrm{dom}(f)$ is a computable set. (That is, we assume that our given structure is computably isomorphic to such a structure $\mathcal{A}$. If not, then we will have already found a counterexample, and our given structure can therefore not be computably categorical, thereby completing the proof.)

We define f by multiples of $(k+m)$ so that $0, \ldots, k-1$ define the k-chains and $k, \ldots, (k+m)-1$

define the m-chains. That is, f is defined as follows for each $n \in \mathbb{N}$:

$$
\begin{aligned}
&k\text{-chains:}\\
f\big((k+m)n\big) &= (k+m)n+1\\
f\big((k+m)n+1\big) &= (k+m)n+2\\
&\vdots\\
f\big((k+m)n+(k-2)\big) &= (k+m)n+(k-1)\\
f\big((k+m)n+(k-1)\big) &= \uparrow
\end{aligned}
$$

$$
\begin{aligned}
&m\text{-chains:}\\
f\big((k+m)n+k\big) &= (k+m)n+(k+1)\\
f\big((k+m)n+(k+1)\big) &= (k+m)n+(k+2)\\
&\vdots\\
f\big((k+m)n+(k+m)-2\big) &= (k+m)n+(k+m)-1\\
f\big((k+m)n+(k+m)-1\big) &= \uparrow
\end{aligned}
$$

Therefore, if we examine which elements of $\mathbb{N}$ do not halt under f, we see that $\overline{\mathrm{dom}(f)} = \{(k+m)n+(k-1) : n \in \mathbb{N}\} \cup \{(k+m)n+(k+m)-1 : n \in \mathbb{N}\}$, which is clearly a computable set.

We again let C be some noncomputable c.e. set with the same restrictions as in Example 2.9. We build $\mathcal{B}$ in much the same manner, modifying the construction so at stage $s+1$ we instead do the following.

- First extend g_s to g_{s+1} by defining g_{s+1} on the following inputs:

  Create a new k-chain:

  $$
  \begin{aligned}
  g_{s+1}\big((k+m)s\big) &\stackrel{\text{def}}{=} (k+m)s+1\\
  g_{s+1}\big((k+m)s+1\big) &\stackrel{\text{def}}{=} (k+m)s+2\\
  &\vdots\\
  g_{s+1}\big((k+m)s+(k+1)\big) &\stackrel{\text{def}}{=} \uparrow
  \end{aligned}
  $$

  Create the first part (k-many) of a new m-chain:

  $$g_{s+1}\big((k+m)s+k\big) \stackrel{\text{def}}{=} (k+m)s+k+1$$

$$\vdots$$

$$g_{s+1}\big((k+m)s+k+(k-2)\big) \stackrel{\text{def}}{=} (k+m)s+k+(k-1)$$

Create the second part ($(m-k)$-many) of a new m-chain:

$$g_{s+1}\big((k+m)s+k+k\big) \stackrel{\text{def}}{=} (k+m)s+k+k+1$$

$$\vdots$$

$$g_{s+1}\big((k+m)s+(k+m)-2\big) \stackrel{\text{def}}{=} (k+m)s+(k+m)-1$$

$$g_{s+1}\big((k+m)s+(k+m)-1\big) \stackrel{\text{def}}{=} \uparrow$$

- Check whether $C_{s+1} - C_s = \emptyset$.

- If $C_{s+1} - C_s \neq \emptyset$, let $i \in C_{s+1} - C_s$. Define g_{s+1} on the following inputs:

  $$g_{s+1}\big((k+m)(i-1)+(k-1)\big) \stackrel{\text{def}}{=} (k+m)s+k+k$$

  $$g_{s+1}\big((k+m)s+k+(k-1)\big) \stackrel{\text{def}}{=} \uparrow$$

  This turns the k-chain corresponding to $n = (i-1)$ into the m-chain corresponding to $n = s$. The first (k-many) part of the m-chain corresponding to $n = s$ then becomes a k-chain.

- If $C_{s+1} - C_s = \emptyset$, further extend g_s to g_{s+1} by defining:

  $$g_{s+1}\big((k+m)s+k+(k-1)\big) \stackrel{\text{def}}{=} (k+m)s+k+k$$

We let $g = \lim_s g_s$, and g is partial computable. As built $\mathcal{B} = (\mathbb{N}, g)$ has infinitely many k-chains and infinitely many m-chains, and is therefore isomorphic to $\mathcal{A}$. We examine which elements of $\mathbb{N}$ do not halt under g. We have that:

$$\begin{aligned} x \notin \text{dom}(g) \iff & x = (k+m)n + (k+m) - 1 \text{ for some } n \in \mathbb{N}, \text{ or,} \\ & x = (k+m)(i-1) + (k-1) \text{ for some } i \notin C, \text{ or,} \\ & x = (k+m)s + k + (k-1) \text{ for some } s \text{ such that } C_{s+1} - C_s \neq \emptyset \end{aligned}$$

Therefore $\overline{\text{dom}(g)} = \{(k+m)n + (k+m) - 1 : n \in \mathbb{N}\} \cup \{(k+m)(i-1) + (k-1) : i \notin C\} \cup \{(k+m)s + k + (k-1) : C_{s+1} - C_s \neq \emptyset\}$. This is essentially the computable join of three sets: $\mathbb{N}$, $\overline{C}$, and $\{s : C_{s+1} - C_s \neq \emptyset\}$, and it is therefore not computable since C is not computable. Therefore, $\mathcal{A}$ and $\mathcal{B}$ cannot be computably isomorphic.

□

We now move on to examine computable categoricity when a partial computable injection structure has infinitely many finite chain orbits of only one single size and additionally an infinite orbit. We show that the existence of even just one infinite orbit will cause computable categoricity to fail.

Theorem 2.10. *If $\mathcal{A}$ is a partial computable injection structure with infinitely many finite chain orbits of only one size (call it ℓ), and one ω-orbit, $\omega*$-orbit, or Z-orbit, then $\mathcal{A}$ is not computably categorical.*

Proof. We first let $\mathcal{A}$ be some partial computable injection structure with infinitely many ℓ-chains (for some $\ell > 0$) and one single ω-orbit. Without loss of generality we assume $\mathcal{A} = (\mathbb{N}, f)$ with f defined as follows so that $\text{dom}(f)$ is a computable set. (That is, we assume that our given structure is computably isomorphic to such a structure $\mathcal{A}$. If not, then we will have already found a counterexample, and our given structure cannot be computably categorical, thereby completing the proof.) We define f by multiples of $(\ell+1)$ so that the "0"s together define the single ω-orbit, and each $1, \ldots, \ell$ defines an ℓ-chain. That is, we define f as follows for each $n \in \mathbb{N}$.

$$\begin{aligned}
\omega\text{-orbit}: \quad & f\big((\ell+1)n\big) = (\ell+1)(n+1) \\
\ell\text{-chains}: \quad & f\big((\ell+1)n+1\big) = (\ell+1)n+2 \\
& f\big((\ell+1)n+2\big) = (\ell+1)n+3 \\
& \vdots \\
& f\big((\ell+1)n+(\ell-1)\big) = (\ell+1)n+\ell \\
& f\big((\ell+1)n+\ell\big) = \uparrow
\end{aligned}$$

Then $\mathcal{A}$ clearly has one ω-orbit and infinitely many ℓ-chains. Furthermore, we see that $\overline{\text{dom}(f)} = \{(\ell+1)n+\ell : n \in \mathbb{N}\}$, which is a computable set. We will computably build $\mathcal{B} = (\mathbb{N}, g)$ up in stages so that $\overline{\text{dom}(g)}$ is not computable.

We again let C be some non-computable, c.e. set enumerated in stages C_s such that: $C = \cup_s C_s$, $C_s \subseteq \{0, \ldots, s-1\}$, and $\text{card}(C_{s+1} - C_s) \leq 1$.

We will build g up in stages, g_s, so that in the end $g = \lim_s g_s$.

stage 0: We begin to build the ω-orbit, and we build one single ℓ-chain. We define g_0 as follows:

$$\begin{aligned}
& g_0\big((\ell+1)\cdot 0\big) = \uparrow \\
& g_0\big((\ell+1)\cdot 0+1\big) = (\ell+1)\cdot 0+2 \\
& g_0\big((\ell+1)\cdot 0+2\big) = (\ell+1)\cdot 0+3
\end{aligned}$$

$$\vdots$$

$$g_0\big((\ell+1)\cdot 0+(\ell-1)\big)=(\ell+1)\cdot 0+\ell$$

$$g_0\big((\ell+1)\cdot 0+\ell\big)=\uparrow$$

stage s: Assume at the end of stage s we have the following:

- $\forall i \leq s$ all of the following hold:

  $$(\ell+1)i \in \mathcal{O}_{g_s}(0)$$

  $$g_s\big((\ell+1)i+1\big)=(\ell+1)i+2$$

  $$\vdots$$

  $$g_s\big((\ell+1)i+(\ell-1)\big)=(\ell+1)i+\ell$$

  $$g_s\big((\ell+1)i+\ell\big)=\uparrow$$

- $\forall i \in C_s$:

  $$(\ell+1)i+\ell \in \mathcal{O}_{g_s}(0)$$

stage s+1: We we now extend g_s to g_{s+1}.

- First extend g_s to g_{s+1} by extending the single ω-orbit by one additional element and building another ℓ-chain. For convenience, we let $a =$ the unique element in $\mathcal{O}_{g_s}(0)$ such that $g_s(a)$ has not yet been defined. Note that if, in the prior stage some i showed up in C_s, then $a=(\ell+1)i+\ell$. If, however, no new i showed up in the prior stage, we will have $a=(\ell+1)s$. We accomplish this by defining g_{s+1} on the following inputs.

  $$g_{s+1}(a)=(\ell+1)(s+1)$$

  $$g_{s+1}\big((\ell+1)(s+1)+1\big)=(\ell+1)(s+1)+1$$

  $$\vdots$$

  $$g_{s+1}\big((\ell+1)(s+1)+(\ell-1)\big)=(\ell+1)(s+1)+\ell$$

  $$g_{s+1}\big((\ell+1)(s+1)+\ell\big)=\uparrow$$

- Now, check whether $C_{s+1}-C_s=\emptyset$.

- If $C_{s+1}-C_s\neq\emptyset$, let i be the unique element in $C_{s+1}-C_s$. Define g_{s+1} on the following:

  $$g_{s+1}\big((\ell+1)(s+1)\big)=(\ell+1)i+1$$

  This has the effect of taking the ℓ-chain built during stage i and attaching it to the end of the ω-orbit. That is $\big((\ell+1)i+1\big)\in O_{g_{s+1}}(0)$, $\ldots$, $\big((\ell+1)i+\ell\big)\in O_{g_{s+1}}(0)$.

- If $C_{s+1}-C_s=\emptyset$, there is nothing further to do and this completes the stage.

Let $g = \lim_s g_s$. This completes the construction. Then g is partial computable, since for any input x we simply carry out the above process until we find a stage s for which $g_s(x) \downarrow$, possibly computing forever if no such stage is reached. Clearly, $\mathcal{B} = (\mathbb{N}, g)$ as built will have one ω-orbit, namely $O_{g_{s+1}}(0)$. Furthermore, since C is not computable, there must be infinitely many $i \notin C$, and hence infinitely many of the ℓ-chains that we've built remain as ℓ-chains and do not at some later point get attached into the ω-orbit. Therefore $\mathcal{B}$ has one ω-orbit and infinitely many ℓ-chains, and hence is isomorphic to $\mathcal{A}$.

If we examine elements that do not halt under g, we find that $\overline{\text{dom}(g)} = \{(\ell+1)n + \ell : n \notin C\}$. Hence $\overline{\text{dom}(g)}$ is not computable since C is not computable. Since $\overline{\text{dom}(f)}$ is computable, this means that $\mathcal{A}$ and $\mathcal{B}$ cannot be computably isomorphic.

For the case where $\mathcal{A}$ has infinitely many ℓ-chains and one ω^*-orbit, we simply modify the proof so that at each stage, we attach elements and ℓ-chains to the *beginning* (left-hand side) of the orbit to create an ω^*-orbit. For the case where $\mathcal{A}$ has infinitely many ℓ-chains and one Z-orbit, we modify the proof so we attach to *both* sides of the orbit (at alternating stages, attach to the beginning/left and then to the end/right), in order to create a Z-orbit.

□

The next case we examine is when our partial computable injection structure $\mathcal{A}$ has infinitely many finite chains of arbitrarily large size. In this instance, we mean that there may not be infinitely many finite chains of any one single size, but there are infinitely many such sizes. We let $\{k_i\}_{i\geq 0}$ represent the sequence of all sizes of finite chain orbits, so that $\mathcal{A}$ has exactly one finite chain orbit of size k_i for each i. Note, then, that with this notation, if $\mathcal{A}$ has, for instance, exactly two m-chains, then m will show up exactly twice in the sequence. For $\mathcal{A}$ to have infinitely many sizes (or equivalently, arbitrarily large sizes), this means that for every $n \in \mathbb{N}$, there exists some i for which $k_i > n$. We break out our examination of such a structure into the following two cases:

(a) $\{k_i\}_{i\geq 0}$ is a computable sequence

(b) $\{k_i\}_{i\geq 0}$ is not a computable sequence

The following result makes significant progress towards showing that partial computable injection structures with finite chains as in case (a), are not computably categorical. We do not examine part (b) here, and instead leave the investigation of this type of structure for future research.. (See Section 3.5 for a slightly lengthier discussion of this topic.)

Theorem 2.11. *Let $\mathcal{A}$ be a partial computable injection structure with infinitely many finite chain orbits of increasingly large arbitrary size. Let $\{k_i\}_{i\geq 0}$ represent the sequence of all sizes of finite chain orbits, so that $\mathcal{A}$ has exactly one finite chain orbit of size k_i for each i. If $\{k_i\}_{i\geq 0}$ is a computable, strictly increasing sequence, then $\mathcal{A}$ is not computably categorical.*

Proof. We let $\mathcal{A} = (\mathbb{N}, f)$ be a partial computable injection structure whose only orbits are finite chains with sizes from our computable, strictly increasing sequence $\{k_i\}_{i\geq 0}$. Without loss of generality, we assume that $\text{dom}(f)$ is a computable set. (That is, we assume that our given structure is computably isomorphic to such a structure $\mathcal{A}$. If not, then we will have already found a counterexample, and our given structure cannot therefore be computably categorical, thereby completing the proof.)

Our goal will be to build $\mathcal{B} = (\mathbb{N}, g)$ to be a partial computable injection structure where $\mathcal{B}$ also has sizes of chains coming from $\{k_i\}_{i\geq 0}$ where $\text{dom}(g)$ is not a computable set. We let C be some noncomputable c.e. set of odd numbers — for example, take $C = \{2e+1 : e \in \text{halting set}\}$. Furthermore, we let C have computable enumeration C_s such that for all s, $C_s \subseteq \{0, 1, \ldots, s-1\}$, $\text{card}(C_{2s+2} - C_{2s+1}) \leq 1$, and $\text{card}(C_{2s+1} - C_{2s}) = 0$. Therefore, C consists only of odd numbers, which show up at only even stages in our enumeration. We will build g up in stages g_s. At each stage s we will add k_s new elements to our structure and we will create a finite chain of size k_s. The way in which we create the finite chains will depend upon our noncomputable set C so that in the end $\text{dom}(g)$ is not computable. For convenience, let $K_i = \sum_{j=0}^{i} k_j$.

stage 0: Create a finite chain of size k_0 starting with 0, as follows:

$$g_0(0) = 1$$
$$g_0(1) = 2$$
$$\vdots$$
$$g_0(k_0 - 2) = k_0 - 1$$
$$g_0(k_0 - 1) = g_0(K_0 - 1) = \uparrow$$

stage s: (Inductive stage) Assume at the end of stage s we have the following:

- g_s consists of exactly one finite chain of size k_i for all $i \leq s$
- $\forall i \leq s$ such that $i \notin C_s$, $g_s(K_i - 1) = \uparrow$
- $\forall i, t \leq s$ such that $i \in C_t - C_{t-1}$, $g_s(K_{t-1} + k_i - 1) = \uparrow$
- $g_s(x) \downarrow$ for all other inputs $x \leq K_s - 1$.

stage s + 1: We now extend g_s to g_{s+1}, and we will add k_{s+1} new elements into the construction at this stage. Note, that at a previous stage $t < s+1$ we may have defined $g_t(x) = \uparrow$, and at this stage we may define $g_{s+1}(x) \downarrow$. This is ok, as we are building g to be a partial computable function, which may not halt on some inputs until we get to a certain stage.

- First check if any new i are enumerated into C_{s+1}.
- If $C_{s+1} - C_s \neq \emptyset$, then let i be the unique element such that $i \in C_{s+1} - C_s$. We create a finite chain of size k_{s+1} by taking the previously defined k_i-chain, adding $(k_{s+1} - k_i)$-many more elements to it, and hence turning it into a k_{s+1}-chain. To accomplish this, we extend g_s as follows:

  $$g_{s+1}(K_i - 1) = K_s + k_i$$
  $$g_{s+1}(K_s + k_i) = K_s + k_i + 1$$
  $$\vdots$$
  $$g_{s+1}(K_s + k_i + (k_{s+1} - k_i) - 2) = K_s + k_i + (k_{s+1} - k_i) - 1$$
  $$g_{s+1}(K_s + k_i + (k_{s+1} - k_i) - 1) = g_{s+1}(K_{s+1} - 1) = \uparrow$$

  Additionally, we replace the k_i-chain that we just turned into a k_{s+1}-chain by extending g_s as follows:

  $$g_{s+1}(K_s) = K_s + 1$$
  $$g_{s+1}(K_s + 1) = K_s + 2$$
  $$\vdots$$
  $$g_{s+1}(K_s + k_i - 2) = K_s + k_i - 1$$
  $$g_{s+1}(K_s + k_i - 1) = \uparrow$$

- If $C_{s+1} - C_s = \emptyset$, then create a new finite chain of size k_{s+1} by defining g_{s+1} on the following inputs.

  $$g_{s+1}(K_s) = K_s + 1$$
  $$g_{s+1}(K_s + 1) = K_s + 2$$
  $$\vdots$$
  $$g_{s+1}(K_s + k_{s+1} - 2) = K_s + k_{s+1} - 1$$
  $$g_{s+1}(K_s + k_{s+1} - 1) = g_{s+1}(K_{s+1} - 1) = \uparrow$$

Now, let $g = \lim_s g_s$. This completes the construction. This limit exists, since once $g_s(x) \downarrow = y$, then $\forall t > s$, we have $g_t(x) = y$. We also have that g is indeed a partial computable function, since to

determine $g(x)$ for some input x, we simply run the above construction until we have specified that $g_s(x) \downarrow$ for some stage s; if we never find such a stage s, we simply compute forever. Note that in this construction, since $\{k_i\}_{i \geq 0}$ is a computable, indeed strictly increasing, sequence, we can always computably determine what the next k_s is. We let $\mathcal{B} = (\mathbb{N}, g)$ and therefore we have that $\mathcal{B}$ is a partial computable injection structure.

Furthermore, $\mathcal{B}$ has exactly one finite chain of size k_i for each $i \geq 0$, which we can see as follows. At each stage s we add exactly one finite chain of size k_s. Sometimes we do this by taking the next k_s elements of $\mathbb{N}$ and turning them into a finite k_s-chain. Sometimes we do this by taking the next k_s elements of $\mathbb{N}$ and attaching $(k_s - k_i)$ of them to a k_i-chain (thus turning the original k_i-chain into a k_s-chain), and creating a new k_i-chain out of the remaining ones in order to replace the original. Since C contains only odd numbers, and those odd numbers show up at only even stages of our enumeration, we know that k_i-chains can only be turned into k_s-chains for even s. Furthermore we know that once a k_i-chain has been turned into a k_s-chain, it remains a k_s-chain thereafter, since no even s will ever show up in our set C. This means there is no danger that one of these original k_i-chains gets added on to infinitely many times, and our construction will indeed yield only finite chains of size k_i for each $i \geq 0$. Therefore, $\mathcal{B}$ is isomorphic to $\mathcal{A}$.

Now, examining which elements do not halt under g — that is, which elements are not in $\mathrm{dom}(g)$ — we have the following:

$$\begin{aligned} x \notin \mathrm{dom}(g) \iff & (x = \text{end of } k_i\text{-chain for some } i \notin C) \ \vee \\ & (x = \text{“mid”-point of } k_s\text{-“chain” where} \\ & \text{new } i \text{ showed up in } C \text{ at stage } s) \\ \iff & (x = K_i - 1 \text{ for some } i \notin C) \ \vee \\ & (x = K_s + k_i - 1 \text{ for some } i \in C_{s+1} - C_s) \end{aligned}$$

$$\therefore \ \overline{\mathrm{dom}(g)} = \{K_i - 1 : i \notin C\} \cup \{K_s + k_i - 1 : i \in C_{s+1} - C_s\}$$

Hence $\overline{\mathrm{dom}(g)}$ is not a computable set since C is not computable. By assumption, in $\mathcal{A}$ $\overline{\mathrm{dom}(f)}$ is a computable set. Therefore $\mathcal{B} = (\mathbb{N}, g)$ cannot be computably isomorphic to $\mathcal{A} = (\mathbb{N}, f)$, and hence $\mathcal{A}$ cannot be computably categorical.

□

We now come back to the case where $\mathcal{A}$ has infinitely many finite chains of one size and infinitely

many cycles of the same or bigger sizes. This situation builds heavily off the ideas in the preceding proofs of Theorem 2.8 and Theorem 2.11. We make progress towards showing that a partial computable injection structure with infinitely many ℓ-chains and infinitely many "$\geq \ell$"-cycles is not computably categorical. Note that unlike in Example 2.9, the cycles do not all have to be of the same size, nor do there have to be infinitely many of any particular size. We simply require that the cycles have size $\geq \ell$ and that there are infinitely many such cycles. For instance, having one $(\ell+1)$-cycle, one $(\ell+2)$-cycle, one $(\ell+3)$-cycle, and so on, would suffice.

We let $\{k_i\}_{i\geq 1}$ represent the sequence of all sizes of "$\geq \ell$"-cycles, so that $\mathcal{A}$ has exactly one cycle of size $k_i \geq \ell$ for each i. Again we note that by this notation we mean that if $\mathcal{A}$ has exactly two m-cycles, then m will show up exactly twice in the sequence. We break out our examination of such a structure into the following two cases:

(a) $\{k_i\}_{i\geq 1}$ can be computably enumerated

(b) $\{k_i\}_{i\geq 1}$ can not be computably enumerated

The following result shows that partial computable injection structures with orbits that consist only of cycles of size $\geq \ell$ as in case (a), are not computably categorical. We do not examine part (b) here, and instead leave the investigation of this type of structure for future research.. (See Section 3.5 for a slightly lengthier discussion of this topic.)

Theorem 2.12. *Let $\mathcal{A}$ be a partial computable injection structure with infinitely many finite chain orbits of size ℓ and infinitely many finite cycles of size $\geq \ell$, and no other types of orbits. Let $\{k_i\}_{i\geq 1}$ represent the sequence of all sizes of finite chain orbits, so that $\mathcal{A}$ has exactly one finite chain orbit of size $k_i \geq \ell$ for each i. If $\{k_i\}_{i\geq 1}$ can be computably enumerated, then $\mathcal{A}$ is not computably categorical.*

Proof. We build heavily off the constructions in Theorem 2.8 and Theorem 2.11. We let $\mathcal{A} = (\mathbb{N}, f)$ be a partial computable injection structure with infinitely many ℓ-chains and with infinitely many cycles of size $\geq \ell$, where the cycle sizes are contained in some sequence $\{k_i\}_{i\geq 1}$ which we can enumerate computably. Furthermore, we assume that $\mathrm{dom}(f)$ is a computable set.

We will build $\mathcal{B} = (\mathbb{N}, g)$ to be a partial computable injection structure with infinitely many ℓ-chains and infinitely many cycles of sizes from $\{k_i\}_{i\geq 1}$ where $k_i \geq \ell$ for each i. To do this, we let C be some noncomputable c.e. set with enumeration C_s such that for all s, $C_s \subseteq \{0, 1, \ldots, s-1\}$ and $\mathrm{card}(C_{s+1} - C_s) \leq 1$. We build g up in stages g_s. At each stage s we will add $(\ell + k_s)$-many new elements to our structure, and we will create an ℓ-chain and a k_s-cycle. The way in which we

create the finite chains and cycles will depend upon our noncomputable set C so that in the end $\mathrm{dom}(g)$ is not computable. For convenience, let $K_i = \sum_{j=1}^{i} k_j$.

stage 0: Define $\forall x,\ g_0(x) \stackrel{\text{def}}{=} \uparrow$.

stage s: (Inductive stage) Assume at the end of stage s we have the following:

- g_s consists of exactly one cycle of size k_i for all $i \leq s$, and exactly s-many ℓ-chains
- $\forall i \leq s$ such that $i \notin C_s$, $g_s(\ell(i-1) + K_{i-1} + \ell - 1) = \uparrow$
- $\forall i, t < s$ such that $i \in C_{t+1} - C_t$, $g_s(\ell t + K_t + \ell + k_{t+1} - 1) = \uparrow$
- $g_s(x) \downarrow$ for all other inputs $x \leq s\ell + K_s - 1$.

stage s + 1: We now extend g_s to g_{s+1}, and we will add $(\ell + k_{s+1})$-many new elements into the construction at this stage.

- First create an ℓ-chain as follows:

$$g_{s+1}\big(\ell s + K_s\big) \stackrel{\text{def}}{=} \ell s + K_s + 1$$
$$\vdots$$
$$g_{s+1}\big((\ell s + K_s) + \ell - 2\big) \stackrel{\text{def}}{=} (\ell s + K_s) + \ell - 1$$
$$g_{s+1}\big((\ell s + K_s) + \ell - 1\big) \stackrel{\text{def}}{=} \uparrow$$

- Now check if any new i are enumerated into C_{s+1}.
- If $C_{s+1} - C_s = \emptyset$, then create a new cycle of size k_{s+1} by defining g_{s+1} on the following inputs.

$$g_{s+1}(\ell s + K_s + \ell) = \ell s + K_s + \ell + 1$$
$$\vdots$$
$$g_{s+1}\big((\ell s + K_s + \ell) + (k_{s+1} - 2)\big)\big) = (\ell s + K_s + \ell) + (k_{s+1} - 1)$$
$$g_{s+1}\big((\ell s + K_s + \ell) + (k_{s+1} - 1)\big)\big) = \ell s + K_s + \ell$$

- If $C_{s+1} - C_s \neq \emptyset$, then let i be the unique element such that $i \in C_{s+1} - C_s$. We take the previously defined ℓ-chain from stage i and turn it into a k_{s+1}-cycle — this will require $(k_{s+1} - \ell)$-many elements. Then we use the remaining ℓ-many elements for this stage to create an ℓ-chain to replace the one that we just turned into a cycle. To accomplish this, we extend g_s as follows:

  Turn ℓ-chain into k_{s+1}-cycle. If $k_{s+1} > \ell + 1$, do all of i.-v. below. If $k_{s+1} = \ell + 1$, do only i. and v. below. If $k_{s+1} = \ell$, do only vi. below.

i. $g_{s+1}\big(\ell(i-1)+K_{i-1}+\ell-1\big)=\ell s+K_s+\ell$

ii. $g_{s+1}(\ell s+K_s+\ell)=\ell s+K_s+\ell+1$

iii. $\vdots$

iv. $g_{s+1}\big((\ell s+K_s+\ell)+(k_{s+1}-\ell-2)\big)=(\ell s+K_s+\ell)+(k_{s+1}-\ell-1)$

v. $g_{s+1}\big((\ell s+K_s+\ell)+(k_{s+1}-\ell-1)\big)=\ell(i-1)+K_{i-1}$

vi. $g_{s+1}\big(\ell(i-1)+K_{i-1}+\ell-1\big)=\ell(i-1)+K_{i-1}$

Create ℓ-chain to replace the one we just turned into a cycle.

$g_{s+1}(\ell s+K_s+\ell)=(\ell s+K_s+\ell)+1$

$\vdots$

$g_{s+1}\big((\ell s+K_s+\ell)+(k_{s+1}-2)\big)=(\ell s+K_s+\ell)+(k_{s+1}-1)$

$g_{s+1}\big((\ell s+K_s+\ell)+(k_{s+1}-1)\big)=(\ell s+K_s+\ell)$

$g_{s+1}(\ell s+K_s+\ell)=\uparrow$

We let $g=\lim_s g_s$. Then by construction, $\mathcal{B}=(\mathbb{N},g)$ is a partial computable injection structure with infinitely many ℓ-chains and infinitely many cycles with sizes from $\{k_1\}_{i\geq 1}$. Therefore $\mathcal{B}\simeq\mathcal{A}$. We now examine elements which are not in $\mathrm{dom}(g)$.

$$x\notin \mathrm{dom}(g) \iff x=\ell(i-1)+K_{i-1}+\ell-1 \text{ for some } i\notin C, \text{ or,}$$
$$x=(\ell s+K_s+\ell)+(k_{s+1}-1) \text{ for some } s \text{ with } C_{s+1}-C_s\neq\emptyset$$

Therefore $\overline{\mathrm{dom}(g)}$ is not a computable set since C is not computable. By assumption, $\overline{\mathrm{dom}(f)}$ is computable. Therefore $\mathcal{A}$ and $\mathcal{B}$ cannot be computably isomorphic. □

This now completes our examination of computable categoricity for the conditions not covered by Corollary 2.6. We have shown that a substantial number of these situations yield non-computably categorical partial injection structures.

Chapter 3

Higher Levels of Categoricity and Index Sets of Partial Injection Structures

We now wish to examine higher levels of categoricity for partial injection structures, specifically Δ^0_2 and Δ^0_3 categoricity. In addition, we also define index sets for partial injection structures, and examine some preliminary results. Again, we are slightly abusing notation here. We say that a partial computable injection structure $\mathcal{A}$ is Δ^0_2 or Δ^0_3 categorical iff every partial computable injection structure isomorphic to $\mathcal{A}$ is isomorphic to $\mathcal{A}$ via an isomorphism which is Δ^0_2 or Δ^0_3 computable. Traditionally, the definitions of Δ^0_2 or Δ^0_3 categorical require that the structures themselves be computable, not just partial computable. As previously mentioned, a straightforward Marker's extension of our partial computable injection structure will yield an equivalent, computable structure.

3.1 Relatively Δ^0_2-Categorical Partial Injection Structures

We begin our examination of higher levels of categoricity of partial injections structures by investigating Δ^0_2-categoricity. Before examining this property for partial injection structures, we look first at injection structures. We take note of the following result about Δ^0_2-categoricity:

Theorem 3.1 (Cenzer, Harizanov, Remmel, [7]). *Let $\mathcal{A} = (A, f)$ be a computable injection structure. Then $\mathcal{A}$ is Δ^0_2-categorical iff $\mathcal{A}$ has finitely many ω-orbits or finitely many Z-orbits.*

We recall that the only types of infinite orbits in injection structures are ω-orbits and Z-orbits. This result allows for possibly infinitely many infinite orbits of one type while maintaining Δ^0_2-categoricity. We recall from our results and discussion throughout Chapter 2 that finite chain orbits often behaved like infinite orbits, since they too had no definitive "end point". We will now show that we get a correspondingly similar result for partial injection structures, once we allow for appropriate consideration and modification of finite chains.

Theorem 3.2. *If $\mathcal{A} = (A, f)$ is a partial computable injection structure which satisfies at least three of the following four conditions:*

- *$\mathcal{A}$ has only finitely many Z-orbits*
- *$\mathcal{A}$ has only finitely many ω-orbits*
- *$\mathcal{A}$ has only finitely many ω^*-orbits*
- *$\mathcal{A}$ has only finitely many sizes of finite chain orbits*

then $\mathcal{A}$ is relatively Δ^0_2-categorical.

Proof. The proof is contained in the following lemmas: Lemma 3.3, Lemma 3.5, Lemma 3.6, and Lemma 3.7. □

In other words, we can allow for exactly one of the four types of (non-cycle) orbits to have possibly infinitely many such orbits or sizes. To show that a partial injection structure requires three of the four conditions to be relatively Δ^0_2-categorical, we breakup the proof and examine the various types of orbits in triples. That is, we examine partial injection structures with the following:

- finitely many Z-orbits, finitely many ω-orbits and finitely many ω^*-orbits (no restriction on finite chain orbits)
- finitely many Z-orbits, finitely many ω-orbits and finite chain orbits of only finitely many sizes (no restriction on ω^*-orbits)
- finitely many Z-orbits, finitely many ω^*-orbits and finite chain orbits of only finitely many sizes (no restriction on ω-orbits)
- finitely many ω-orbits, finitely many ω^*-orbits, and finite chain orbits of only finitely many sizes (no restriction on Z-orbits)

Note that in all of the above cases, as with Theorem 2.4 and Theorem 2.5, we do not place any restriction on cycle orbits. That is, we can allow for possibly infinitely many cycles without harming Δ^0_2-categoricity. We proceed now with the proof for the first triple.

Lemma 3.3. *If $\mathcal{A} = (A, f)$ is a partial computable injection structure with finitely many Z-orbits, finitely many ω-orbits and finitely many ω^*-orbits, then $\mathcal{A}$ is relatively Δ^0_2-categorical.*

Proof. This proof follows similar methods as those used in Theorem 2.4 and Theorem 2.5. Let $\mathcal{A} = (A, f)$ be a partial computable injection structures with finitely many Z-orbits, ω-orbits, and ω^*-orbits as follows:

- m ω-orbits with representatives $a_1, \ldots, a_m$ s.t. $a_i \notin \text{range}(f)$
- n ω^*-orbits with representatives $b_1, \ldots, b_n$ s.t. $b_i \notin \text{dom}(f)$
- l Z-orbits with representatives $z_1, \ldots, z_l$ s.t. $\mathcal{O}_f(z_i) \neq \mathcal{O}_f(z_j)$ for $i \neq j$

Goncharov ([22]) proved that $\mathcal{A}$ is relatively computably categorical iff $\mathcal{A}$ has a formally c.e. Scott family, and Ash ([2]) extended this notion to Δ^0_α-categoricity. Therefore, to show that $\mathcal{A}$ is Δ^0_2-categorical, we will show that $\mathcal{A}$ has a Σ^0_2 Scott family, consisting of computable Σ^0_2 formulas.

Let $\vec{d} = d_0, \ldots, d_r$ be a sequence in A distinct from $a_1, \ldots, a_m$, from $b_1, \ldots b_n$, and from $z_1, \ldots, z_l$. Then $\forall q \leq r$ we have that either d_q is part of a cycle, a chain, an ω-orbit, an ω^*-orbit, or a Z-orbit. This corresponds to one of the following:

(1) $\exists t\, f^k_t(d_q) = d_q$ (for some fixed, minimal $k > 0$)

(2) $\exists a \exists t_1 \exists t_2 \forall b \forall s \left[\left(f^k_s(a) \uparrow\right) \wedge \left(f^{k-1}_{t_1}(a) \downarrow\right) \wedge \left(f_s(b) \neq a\right) \wedge \left(f^c_{t_2}(a) = d_q\right) \right]$

(for some fixed $k > 0, c < k$)

(3) $\exists t\, f^k_t(a_i) = d_q$ (for some fixed $k \geq 0, i \in \{1, \ldots, m\}$)

(4) $\exists t\, f^k_t(d_q) = b_i$ (for some fixed $k \geq 0, i \in \{1, \ldots, n\}$)

(5) $\exists t\, f^k_t(d_q) = z_i$ (for some fixed $k \geq 0, i \in \{1, \ldots, l\}$)

(6) $\exists t\, f^k_t(z_i) = d_q$ (for some fixed $k > 0, i \in \{1, \ldots, l\}$)

Similar to Theorem 2.4 and Theorem 2.5, we note that in (1) if we have more than one element in our sequence $\vec{d}$ which meets this condition for the same value of k, then we have to specify whether they are in the same cycle or different cycles. Therefore for d_q, d_Q in our sequence $\vec{d}$ with $q \neq Q$ and $q, Q \leq r$, if both d_q and d_Q satisfy the same formula (1) with the same value of k then either:

(7) $\exists t\, f_t^j(d_q) \downarrow = d_Q$ (for some fixed $j < k$)

(8) $\forall j < k\ \exists t\, f_t^j(d_q) \downarrow \neq d_Q$

Similarly, if we have more than one element, d_q and d_Q, in our sequence $\vec{d}$ which satisfies condition (2) for the same value of k, regardless of the value of c, then we must specify whether they are in the same or different finite chains. For convenience, we let c_q represent the value of c from statement (2) for d_q, and we let c_Q represent the value of c from statement (2) for d_Q. Without loss of generality, we assume that $c_q \leq c_Q$. We have the following:

(9) $\exists t\, f_t^{c_Q - c_q}(d_q) = d_Q$

(10) $\forall t\ f_t^{c_Q - c_q}(d_q) \neq d_Q$

It is clear that (2) is a Σ_2^0 statement, that (10) is a Π_1^0 statement, and that (3)-(7) and (9) are all Σ_1^0 statements since f is a partial computable function. We can expand (1) and (8) in the same manner we did in Theorem 2.4 to show that they are also Σ_1^0 statements.

To create the Σ_2^0 Scott family, then, we simply take appropriate conjunctions of (1)-(10). As defined, this is clearly a countable family. To complete the proof, we need only show that given two sequences, $\vec{d}$ and $\vec{e}$, in A, if they satisfy the same Scott formula, then there exists an automorphism, h, such that $h(\vec{d}) = \vec{e}$. We first define a partial function h as follows:

$$h(x) = \begin{cases} x & \text{if } x \text{ is in an infinite orbit} \\ x & \text{if } x \text{ is in a cycle or finite chain distinct from } \vec{d} \text{ and } \vec{e} \\ f^j(e_q) & \text{if } x \text{ is in a } k\text{-cycle and } x = f^j(d_q) \text{ for some } j < k \\ f^j(e_q) & \text{if } x \text{ is in finite chain and } x = f^j(d_q) \text{ for some } j < k - c \\ & \quad \text{where } k, c \text{ are as in (2)} \\ y \text{ s.t. } f^j(y) = e_q & \text{if } x \text{ is in finite chain and } f^j(x) = d_q \text{ for some } 0 < j < c \\ & \quad \text{where } c \text{ is as in (2)} \end{cases}$$

It is clear then that if x is in an infinite orbit or if x is in a cycle or finite chain distinct from $\vec{d}$ and $\vec{e}$, then h is the identity function, and therefore is 1-1 and preserves these orbits. For x in one of the cycles or finite chains that contains one of $d_1, \ldots, d_r$, it is easy to see that the Scott formulas for $\vec{d}$ and $\vec{e}$ from (1), (2), (7) or (8), and (9) or (10) as appropriate, will ensure that the orbits are preserved under h, and that h is 1-1. Finally, then, we extend h from a partial function to a total

one in the same manner as we did for Theorem 2.5 (taking cycles to cycles and chains to chains). This will yield our desired automorphism.

□

Before showing that a partial computable injection structure with finitely many Z-orbits and finitely many ω-orbits is relatively Δ^0_2 categorical if it also has finitely many sizes of finite chain orbits, we first show this holds if it has finitely many finite chain orbits.

Lemma 3.4. *If $\mathcal{A} = (A, f)$ is a partial computable injection structure with finitely many Z-orbits, finitely many ω-orbits and finitely many finite chain orbits, then $\mathcal{A}$ is relatively Δ^0_2-categorical.*

Proof. The proof of this follows closely that of Lemma 3.3, with some slight modifications. Let $\mathcal{A} = (A, f)$ be a partial computable injection structures with finitely many Z-orbits, ω-orbits, and finite chain-orbits as follows:

- m ω-orbits with representatives $a_1, \ldots, a_m$ s.t. $a_i \notin \text{range}(f)$
- l Z-orbits with representatives $z_1, \ldots, z_l$ s.t. $\mathcal{O}_f(z_i) \neq \mathcal{O}_f(z_j)$ for $i \neq j$
- p finite chain-orbits with representatives $c_1, \ldots, c_p$ s.t. $c_i \notin \text{dom}(f)$

To create our Scott family, we take (1), (3), and (5)-(8) to be the same as in Lemma 3.3. We replace (2) and (4) as follows:

(2) $\exists t\, f^k_t(d_q) \downarrow = c_i$ (for some fixed $k \geq 0, i \in \{1, \ldots, p\}$)

(4) $\exists s \forall t\, [(f^{k-1}_s(d_q) \downarrow \neq c_i) \wedge (f^k_t(d_q) \uparrow)]$ (for some fixed $k > 0$, and $\forall i \in \{1, \ldots, p\}$)

Additionally, if we have more than one element, d_q and d_Q, in our sequence $\vec{d}$ which satisfies condition (4) regardless of the value of k, then we must specify whether they are in the same or different ω^*-orbits. For convenience, we let k_q represent the value of k from statement (4) for d_q, and we let k_Q represent the value of k from statement (4) for d_Q. Without loss of generality, we assume that $k_q \geq k_Q$. We have the following:

(9) $\exists t\, f^{k_q - k_Q}_t(d_q) = d_Q$

(10) $\forall t\, f^{k_q - k_Q}_t(d_q) \neq d_Q$

It is clear that (2) and (9) are Σ^0_1 statements, (10) is a Π^0_1 statement, and (4) is a Σ^0_2 statement, since f is partial computable. Then the Σ^0_2 Scott family for $\mathcal{A}$ consists of appropriate conjunctions of (1)-(10). As defined, this is clearly a countable family. To complete the proof, we need only show

that given two sequences, $\vec{d}$ and $\vec{e}$, in A, if they satisfy the same Scott formula, then there exists an automorphism, h, such that $h(\vec{d}) = \vec{e}$. We first define a partial function h as follows.

$$h(x) = \begin{cases} x & \text{if } x \text{ is in } Z\text{-orbit, } \omega\text{-orbit, or finite chain} \\ x & \text{if } x \text{ is in a cycle or } \omega^*\text{-orbit distinct from } \vec{d} \text{ and } \vec{e} \\ f^j(e_q) & \text{if } x \text{ is in a } k\text{-cycle and } x = f^j(d_q) \text{ for some } j < k \\ f^j(e_q) & \text{if } x \text{ is in } \omega^*\text{-orbit, and } x = f^j(d_q) \text{ for some } j \leq k \\ & \quad \text{where } k \text{ is as in (4)} \\ y \text{ s.t. } f^j(y) = e_q & \text{if } x \text{ is in } \omega^*\text{-orbit, and } f^j(x) = d_q \text{ for some } j > 0 \end{cases}$$

We now extend h from a partial function to a total one in a similar manner as we did for Theorem 2.5 and Lemma 3.3. We take cycles to cycles, with $h(f^j(e_q)) = f^j(d_Q)$ for the next smallest q and Q with d_Q not yet in range(h). We also take ω^*-orbits to ω^*-orbits, sending the ω^*-orbit containing the smallest q for which e_q is not yet mapped in h to the ω^*-orbit containing the smallest Q for which d_Q is not yet in range(h), ensuring that we map the ends (right most points) of each ω^*-orbit accordingly. Then, as before, it is easy to see that h now defines a total function which is 1-1, onto, and preserves orbits, and therefore we have defined our desired automorphism. □

We now take the previous result even one step further and allow for possibily infinitely many finite chain orbits, as long as those finite chains are of only finitely many sizes. The key concept in this proof is that even though we allow for possibly infinitely many finite chains, as long as the sizes of the finite chains have some finite bound on them we can distinguish a finite chain from an infinite orbit (in this case an ω^*-orbit) because we know exactly when to "stop waiting".

Lemma 3.5. *If $\mathcal{A} = (A, f)$ is a partial computable injection structure with finitely many Z-orbits, finitely many ω-orbits and finite chain orbits of only finitely many sizes, then $\mathcal{A}$ is relatively Δ^0_2-categorical.*

Proof. The proof of this follows closely that of Lemma 3.4, with some slight modifications. Let $\mathcal{A} = (A, f)$ be a partial computable injection structure with finitely many Z-orbits and ω-orbits, and finite chain-orbits as follows:

- m ω-orbits with representatives $a_1, \ldots, a_m$ s.t. $a_i \notin \text{range}(f)$
- l Z-orbits with representatives $z_1, \ldots, z_l$ s.t. $\mathcal{O}_f(z_i) \neq \mathcal{O}_f(z_j)$ for $i \neq j$

- finite chain-orbits only of sizes $\ell_1, \ldots, \ell_p$

To create our Scott family, we take (1), (3), and (5)-(8) to be the same as in Theorem 2.4. To replace (2) and (4) we note the following. Let $\ell = \max\{\ell_1, \ldots, \ell_p\}$.

$$d_q \in \text{ finite chain } \iff d_q \text{ in finite chain of size } \ell_i \text{ for some } i \in \{1, \ldots, p\}$$

$$d_q \in \omega^*\text{-orbit} \iff (d_q \text{ in an orbit which does not go infinitely forward }) \wedge$$
$$(\text{the orbit goes backward farther than the largest } \ell_i)$$

We can now replace (2) and (4) as follows:

(2) $\exists a \exists t_1 \exists t_2 \forall s \forall b \left[\left(f_s(b) \neq a \right) \wedge \left(f_s^{\ell_i}(a) \uparrow \right) \wedge \left(f_{t_1}^{\ell_i - 1}(a) \downarrow \right) \wedge \left(f_{t_2}^{k}(a) = d_q \right) \right]$

(for some fixed $i \in \{1, \ldots, p\}$ and $k < \ell_i$)

(4) $\exists b \exists t_1 \exists a \exists t_2 \forall s \left[\left(f_s(b) \uparrow \right) \wedge \left(f_{t_1}^{k}(d_q) = b \right) \wedge \left(f_{t_2}^{\ell}(a) = b \right) \right]$

(for some fixed $k > 0$)

Note, if we have more than one element, d_q and d_Q, in our sequence $\vec{d}$ which satisfies condition (2), for the same value of ℓ_i, regardless of the value of k, then we must specify whether they are in the same or different finite chains. For convenience, we let k_q represent the value of k from statement (2) for d_q, and we let k_Q represent the value of k from statement (2) for d_Q. Without loss of generality, we assume that $k_q \leq k_Q$. We have the following:

(9) $\exists t \, f_t^{k_Q - k_q}(d_q) = d_Q$

(10) $\forall t \, f_t^{k_Q - k_q}(d_q) \neq d_Q$

Additionally, if we have more than one element, d_q and d_Q, in our sequence $\vec{d}$ which satisfies condition (4), regardless of the value of k, then we must specify whether they are in the same or different ω^*-orbits. Let k_q represent the value of k from statement (4) for d_q, and let k_Q represent the value of k from statement (4) for d_Q, and without loss of generality, assume that $k_q \geq k_Q$. We have the following:

(11) $\exists t \, f_t^{k_q - k_Q}(d_q) = d_Q$

(12) $\forall t \, f_t^{k_q - k_Q}(d_q) \neq d_Q$

It is clear that (2) and (4) as written are Σ_2^0 statements. We know from Theorem 2.4 that statements (1), (3), and (5) - (8) are all Σ_1^0 statements, and hence are also Σ_2^0 statements. Examining

(9)-(12), we see that (9) and (11) are Σ^0_1 statements, and (10) and (12) are Π^0_1 statements, hence they are all also Σ^0_2 statements. Then the Σ^0_2 Scott family for $\mathcal{A}$ consists of appropriate conjunctions of (1)-(12). As defined, this is clearly a countable family. To complete the proof, we need only show that given two sequences, $\vec{d}$ and $\vec{e}$, in A, if they satisfy the same Scott formula, then there exists an automorphism, h, such that $h(\vec{d}) = \vec{e}$. Below we begin by defining a partial function, h.

$$h(x) = \begin{cases} x & \text{if } x \text{ is in } Z\text{-orbit, } \omega\text{-orbit} \\ x & \text{if } x \text{ is in cycle or } \omega^*\text{-orbit, or finite chain distinct from } \vec{d} \text{ and } \vec{e} \\ f^j(e_q) & \text{if } x \text{ is in a } k\text{-cycle and } x = f^j(d_q) \text{ for some } j < k \\ f^j(e_q) & \text{if } x \text{ is in finite chain and } x = f^j(d_q) \text{ for some } j < \ell_i - k \\ & \quad \text{where } \ell_i, k \text{ are as in (2)} \\ y \text{ s.t. } f^j(y) = e_q & \text{if } x \text{ is in finite chain and } f^j(x) = d_q \text{ for some } 0 < j < k \\ & \quad \text{where } k \text{ is as in (2)} \\ f^j(e_q) & \text{if } x \text{ is in } \omega^*\text{-orbit, and } x = f^j(d_q) \text{ for some } j \leq k \\ & \quad \text{where } k \text{ is as in (4)} \\ y \text{ s.t. } f^j(y) = e_q & \text{if } x \text{ is in } \omega^*\text{-orbit, and } f^j(x) = d_q \text{ for some } j > 0 \end{cases}$$

We then extend h in the same way as we did in Lemma 3.3 and Lemma 3.4. As before, it is easy to see that h then defines the desired automorphism.

□

Now that we have done the initial setup work, we can continue on to prove the remaining two cases of relative Δ^0_2-categoricity. In the first, we allow for possibly infinitely many ω-orbits. And in the final, we allow for possibly infinitely many Z-orbits.

Lemma 3.6. *If $\mathcal{A} = (A, f)$ is a partial computable injection structure with finitely many Z-orbits, finitely many ω^*-orbits and finite chain orbits of only finitely many sizes, then $\mathcal{A}$ is relatively Δ^0_2-categorical.*

Proof. Again, the proof follows closely that of Lemma 3.5, with some slight modifications. Let $\mathcal{A} = (A, f)$ be a partial computable injection structure with finitely many Z-orbits and ω^*-orbits, and finite chain-orbits as follows:

- n ω^*-orbits with representatives $b_1, \ldots, b_n$ s.t. $b_i \notin \text{dom}(f)$

- l Z-orbits with representatives $z_1, \ldots, z_l$ s.t. $\mathcal{O}_f(z_i) \neq \mathcal{O}_f(z_j)$ for $i \neq j$
- finite chain-orbits only of sizes $\ell_1, \ldots, \ell_p$

To create our Scott family, we take (1) and (4)-(8) to be the same as in Theorem 2.4. We take (2), (9), and (10) to be the same as in Lemma 3.5. We replace (3), (11), and (12) as follows, assuming $k_q \leq k_Q$.

(3) $\exists a \exists t_1 \exists t_2 \forall b \forall s \left[\left(f_s(b) \neq a \right) \wedge \left(f_{t_2}^{\ell}(a) \downarrow \right) \wedge \left(f_{t_1}^{k}(a) = d_q \right) \right]$ (for some fixed $k \geq 0$)

(11) $\exists t \, f_t^{k_Q - k_q}(d_q) = d_Q$

(12) $\forall t \, f_t^{k_Q - k_q}(d_q) \neq d_Q$

We construct h similarly and the rest follows as before.

□

Lemma 3.7. *If $\mathcal{A} = (A, f)$ is a partial computable injection structure with finitely many ω-orbits, finitely many ω^*-orbits, and finite chain orbits of only finitely many sizes, then $\mathcal{A}$ is relatively Δ_2^0-categorical.*

Proof. Let $\mathcal{A} = (A, f)$ be a partial computable injection structure with finitely many ω-orbits and ω^*-orbits, and finite chain-orbits as follows:

- m ω-orbits with representatives $a_1, \ldots, a_m$ s.t. $a_i \notin \text{range}(f)$
- n ω^*-orbits with representatives $b_1, \ldots, b_n$ s.t. $b_i \notin \text{dom}(f)$
- finite chain-orbits only of sizes $\ell_1, \ldots, \ell_p$

To create our Scott family, we take (1), (3), (4), (7), and (8) to be the same as in Theorem 2.4. We take (2), (9), and (10) to be the same as in Lemma 3.5. We replace (5) as follows.

(5) $\exists t \forall k \forall s \left[\left(f_t^{\ell}(d_q) \downarrow \right) \wedge \left(f_s^{k}(a_i) \neq d_q \right) \wedge \left(f_s^{k}(d_q) \neq b_j \right) \right]$ (for all $i \in \{1, \ldots, m\}$, $j \in \{1, \ldots, n\}$)

Now in (5) if we have more than one element in our sequence $\vec{d}$ which meets this condition, we must specify whether they are in the same Z-orbit, or different Z-orbits. This is accomplished by replacing (6) as follows, and adding a new formula (11).

(6) $\exists t \, f_t^{j}(d_q) = d_Q$ (for some fixed $j \in \mathbb{N}$)

(11) $\forall l \forall t \left(f_t^{l}(d_q) \neq d_Q \wedge f_t^{l}(d_Q) \neq d_q \right)$

The Σ^0_2 Scott family then consists of appropriate combinations of (1)-(11). We construct h similarly and the rest follows as before.

□

This concludes the proof of Theorem 3.2. Since we know that relative Δ^0_2-categoricity implies Δ^0_2-categoricity, we have the following corollary.

Corollary 3.8. *If $\mathcal{A} = (A, f)$ is a partial computable injection structure with at least three of the following four types of orbits:*

- *finitely many Z-orbits*
- *finitely many ω-orbits*
- *finitely many ω^*-orbits*
- *finite chain orbits of finitely many sizes*

then $\mathcal{A}$ is Δ^0_2-categorical.

We now show that if we restrict ourselves to partial injection structures which consist of only ω-orbits, ω^*-orbits, and cycles, these structures are also relatively Δ^0_2-categorical.

Theorem 3.9. *If $\mathcal{A} = (A, f)$ is a partial computable injection structure with no Z-orbits and no finite chain orbits, then $\mathcal{A}$ is relatively Δ^0_2-categorical.*

Proof. Let $\mathcal{A} = (A, f)$ be a partial computable injection structure with no Z-orbits and no finite chains. That is, $\mathcal{A}$ consists only of cycles, ω-orbits, and ω^*-orbits. We again build a Σ^0_2 Scott family to show that $\mathcal{A}$ is relatively Δ^0_2-categorical. To do so, let $\vec{d} = d_0, \ldots, d_r$ be a sequence in A. Then, $\forall q \leq r$, d_q is either part of a cycle, an ω-orbit, or an ω^*-orbit. This corresponds to one of the following:

(1) $\exists t\, f^k_t(d_q) = d_q$ (for some fixed, minimal $k > 0$)

(2) $\exists a \exists s \forall b \forall t\, f_t(b) \neq a \wedge f^k_s(a) = d_q$ (for some fixed $k \geq 0$)

(3) $\exists b \exists s \forall t\, f_t(b) \uparrow \wedge f^k_s(d_q) = b$ (for some fixed $k \geq 0$)

We note that if we have more than one element, d_q and d_Q, in our sequence $\vec{d}$ which satisfies condition (1) for the same value of k, then we must specify whether they are in the same or different cycles. This is accomplished with the following:

(4) $\exists t\, f_t^j(d_q) = d_Q$ (for some fixed $j < k$)

(5) $\forall j < k\, \exists t\, f_t^j(d_q) \downarrow \neq d_Q$

Similarly, if we have more than one element, d_q and d_Q, in our sequence $\vec{d}$ which satisfies condition (2), regardless of the value of k, then we must specify whether they are in the same or different ω-orbits. For convenience, we let k_q represent the value of k from statement (2) for d_q, and we let k_Q represent the value of k from statement (2) for d_Q. Without loss of generality, we assume that $k_q \leq k_Q$. We have the following:

(6) $\exists t\, f_t^{k_Q - k_q}(d_q) = d_Q$

(7) $\forall t\, f_t^{k_Q - k_q}(d_q) \neq d_Q$

Finally, if we have more than one element, d_q and d_Q, in our sequence $\vec{d}$ which satisfies condition (3), regardless of the value of k, then we must again specify whether they are in the same or different ω^*-orbits. We again let k_q represent the value of k from statement (3) for d_q and k_Q represent the value of k from statement (3) for d_Q. Without loss of generality, we assume that $k_q \geq k_Q$. We have the following:

(8) $\exists t\, f_t^{k_q - k_Q}(d_q) = d_Q$

(9) $\forall t\, f_t^{k_q - k_Q}(d_q) \neq d_Q$

We take our Scott family for $\mathcal{A}$ to be appropriate conjunctions of (1)-(9). As defined, this is clearly a countable family. Additionally, it is clear that (1)-(9) are all Σ_2^0 statements (expanding (1) as in Theorem 2.4). Therefore we have a Σ_2^0 Scott family. We now need only show that given two sequences $\vec{d}$ and $\vec{e}$ in A that satisfy the same Scott sentence, then there is an automorphism h which takes $\vec{d}$ to $\vec{e}$. This is accomplished by first defining a partial function, h, as follows:

$$h(x) = \begin{cases} x & \text{if } x \text{ is in cycle, or } \omega\text{-orbit, or } \omega^*\text{-orbit distinct from } \vec{d} \text{ and } \vec{e} \\ f^j(e_q) & \text{if } x \text{ is in a } k\text{-cycle and } x = f^j(d_q) \text{ for some } j < k \\ f^j(e_q) & \text{if } x \text{ is in } \omega\text{-orbit, and } x = f^j(d_q) \text{ for some } j \geq k \ (k \text{ is as in (2)}) \\ y \text{ s.t. } f^j(y) = e_q & \text{if } x \text{ is in } \omega\text{-orbit, and } f^j(x) = d_q \text{ for some } j < k (k \text{ is as in (2)}) \\ f^j(e_q) & \text{if } x \text{ is in } \omega^*\text{-orbit, and } x = f^j(d_q) \text{ for some } j \leq k \ (k \text{ is as in (3)}) \\ y \text{ s.t. } f^j(y) = e_q & \text{if } x \text{ is in } \omega^*\text{-orbit, and } f^j(x) = d_q \text{ for some } j > 0 \end{cases}$$

We then extend h in a similar manner as in Lemma 3.5, taking this time cycles to cycles, ω^*-orbits to ω^*-orbits, and ω-orbits to ω-orbits, ensuring that we map the beginnings/ends of the ω- and ω^*-orbits appropriately. Then, as before it is easy to see that once extended, h defines the required automorphism. □

Since relative Δ^0_2-categoricity implies Δ^0_2-categoricity, the following corollary is immediate.

Corollary 3.10. *If $\mathcal{A} = (A, f)$ is a partial computable injection structure with no Z-orbits and no finite chain orbits, then $\mathcal{A}$ is relatively Δ^0_2-categorical.*

We now have a quite useful classification of types of partial injection structures which are Δ^0_2-categorical. The next natural question is: are these the only types of partial injection structures which are Δ^0_2-categorical? In the following section, we examine what happens in the other cases.

3.2 Non-Δ^0_2-Categorical Partial Injection Structures

In this section, we explore types of partial injection structures which are not Δ^0_2-categorical. In particular, we investigate the partial injection structures which were not covered by Theorems 3.2 and 3.9. The cases which we have not yet discussed are:

- Infinitely many Z-orbits and ...
  - infinitely many ω-orbits, or
  - infinitely many ω^*-orbits
- Infinitely many sizes of finite chains and ...
  - infinitely many ω^*-orbits, or
  - infinitely many ω-orbits, or
  - infinitely many Z-orbits

We begin by noting the work done by Cenzer, Harizanov, and Remmel on Δ^0_2-categoricity of computable injection structures. As part of proving Theorem 3.1, they obtained the following result.

Theorem 3.11 (Cenzer Harizanov, and Remmel, [7])**.** *Let $\mathcal{A} = (A, f)$ be a computable injection structure with infinitely many ω-orbits and infinitely many Z-orbits. Then $\mathcal{A}$ is not Δ^0_2-categorical.*

Since ω-orbits and Z-orbits are the only types of infinite orbits in injection structures, we easily extend the theorem to account for ω^*-orbits in partial injection structures.

Theorem 3.12. *If $\mathcal{A}$ is a partial computable injection structure with infinitely many Z-orbits and either infinitely many ω-orbits or infinitely many ω^*-orbits and no other types of orbits, then $\mathcal{A}$ is not Δ^0_2-categorical.*

Proof. For infinitely many Z- and ω-orbits, the proof follows the same as in [7]. For infinitely many Z- and ω^*-orbits, we simply modify the construction in the proof given in [7]. Instead of attaching to the ends of the orbits to create ω-orbits, we instead attach to the beginnings of the orbits to create ω^*-orbits. □

We now move on to examine how Δ^0_2-categoricity is affected by finite chain orbits. Specifically when our partial injection structure has infinitely many sizes of finite chains. We make use of the following lemma in our constructions. Note that the concepts presented here are mostly standard (see [42], for instance), with one small addition which will be necessary for our later proofs.

Lemma 3.13. *Let C be some Σ^0_2 set. Then there exists a total computable function $F : \mathbb{N} \times \mathbb{N} \to \{0, 1\}$ such that $\forall x\ \{s : F(x, s) = 0\}$ is infinite, and such that:*

$$x \in C \iff \{s : F(x, s) = 1\} \text{ is finite}$$

Proof. Let C be some Σ^0_2 set. First, we note that the set $\text{Fin} \stackrel{\text{defn}}{=} \{e : W_e \text{ is finite}\}$ is well-known to be Σ^0_2-complete (see [42]). Therefore $C \leq_1 \text{Fin}$, and there exists a 1-reduction, $\tilde{f} : C \to \text{Fin}$ such that:

$$x \in C \iff \tilde{f}(x) \in \text{Fin} \iff W_{\tilde{f}(x)} \text{ is finite}$$

For a given x, using standard convention (see [42]) we can enumerate $W_{\tilde{f}(x)}$ in stages so that at each finite stage, s:

- $\{0, 1, \ldots, s-1\} \subseteq W_{\tilde{f}(x),s}$
- $W_{\tilde{f}(x),s} \subseteq W_{\tilde{f}(x),s+1}$, and
- $\text{card}(W_{\tilde{f}(x),s+1} - W_{\tilde{f}(x),s}) \leq 1$

For our purposes, we wish to alter this standard convention just slightly. This is the one addition which will be necessary for our later proofs. We wish to replace the final condition, and instead of enumerating at most one new element into $W_{\tilde{f}(x),s}$ at *every* stage, s, we wish to enumerate at most one new element into $W_{\tilde{f}(x),s}$ at *every other* stage, s. We replace the final condition with the following:

- $\text{card}(W_{\tilde{f}(x),s+2} - W_{\tilde{f}(x),s}) \leq 1 \qquad (\star)$

We now complete the details of the proof. We note that we add an element to $W_{\tilde{f}(x)}$ at stage $s+1 \iff W_{\tilde{f}(x),s} \neq W_{\tilde{f}(x),s+1}$. And we also note that our final (new) condition gives us that $W_{\tilde{f}(x),s} \neq W_{\tilde{f}(x),s+1} \implies W_{\tilde{f}(x),s+1} = W_{\tilde{f}(x),s+2}$. We now define the function $F : \mathbb{N} \times \mathbb{N} \to \{0,1\}$ as follows:

$$F(x,s) = \begin{cases} 0 & \text{if } W_{\tilde{f}(x),s} = W_{\tilde{f}(x),s+1} \\ 1 & \text{if } W_{\tilde{f}(x),s} \neq W_{\tilde{f}(x),s+1} \end{cases}$$

Since we add at most one new element into $W_{\tilde{f}(x),s}$ at *every other* stage, s, this means that there are infinitely many stages, s, for which we do not add any new elements to $W_{\tilde{f}(x)}$. Specifically, there are infinitely many stages, s, for which $W_{\tilde{f}(x),s} = W_{\tilde{f}(x),s+1}$, and hence there are infinitely many s for which $F(x,s) = 0$. Therefore, for any given x, $\{s : F(x,s) = 0\}$ is infinite.

Additionally, for any e, we know that W_e finite $\iff$ there are only finitely many stages, s, where we add only one element to $W_{\tilde{f}(x)} \iff$ only finitely many s for which $W_{e,s} \neq W_{e,s+1}$. Therefore, applying this back to our function F and our set C gives us:

$$\begin{aligned} x \in C &\iff \tilde{f}(x) \in \text{ Fin } \iff W_{\tilde{f}(x)} \text{ finite} \\ &\iff \text{ finitely many } s \text{ for which } W_{\tilde{f}(x),s} \neq W_{\tilde{f}(x),s+1} \\ &\iff \text{finitely many } s \text{ for which } F(x,s) = 1 \\ &\iff \{s : F(x,s) = 1\} \text{ is finite} \end{aligned}$$

Finally, we note that the function F is indeed total computable. Given x and s, we first simply calculate $\tilde{f}(x)$. This is a total computable function since $\tilde{f}$ is a 1-reduction. Then we find $W_{\tilde{f}(x)}$ and run the enumeration of $W_{\tilde{f}(x)}$ for $s+1$ steps. We know (see [42]) that finding $W_{\tilde{f}(x)}$ and running it's enumeration for a fixed number of steps is a computable process that finishes in a finite amount of time. Then we check whether $W_{\tilde{f}(x),s} = W_{\tilde{f}(x),s+1}$ or $W_{\tilde{f}(x),s} \neq W_{\tilde{f}(x),s+1}$, and we output 0 or 1 accordingly. Therefore F is a total computable function, as desired.

□

We are now ready to explore partial computable injection structures with infinitely many sizes of finite chain orbits. We begin first by examining the case where our partial injection structure $\mathcal{A}$ has infinitely many n-chains for every natural number n. (That is, $\mathcal{A}$ has infinitely many 1-chains, and infinitely many 2-chains, and infinitely many 3-chains, and so on.)

Theorem 3.14. *Let $\mathcal{A} = (A, f)$ be a partial computable injection structure with infinitely many ω^*-orbits, infinitely many finite chains of every size, and no other types of orbits. Then $\mathcal{A}$ is not Δ^0_2-categorical.*

Proof. First, we assume that $\mathcal{A} = (\mathbb{N} - \{0\}, f)$, and that $\mathcal{A}$ has infinitely many ω^*-orbits, infinitely many finite chains of every size, no other types of orbits. Additionally, we assume that the set of all elements in ω^*-orbits is computable. (If not, we assume that our given structure is Δ^0_2-isomorphic to some such $\mathcal{A}$, and proceed. Note that if this is not the case, then we are done – our structure cannot be Δ^0_2-categorical.)

Our goal will be to build some partial computable injection structure $\mathcal{B} = (\mathbb{N} - \{0\}, g)$ with infinitely many ω^*-orbits and infinitely many finite chains of every size, but in which the set of all elements in ω^*-orbits under g is not Δ^0_2. This will give us that $\mathcal{B} \simeq \mathcal{A}$, but $\mathcal{B}$ is not Δ^0_2-isomorphic to $\mathcal{A}$ (since if it were we would have a Δ^0_2-function applied to a computable set, yielding an isomorphic set that is not Δ^0_2). Hence $\mathcal{A}$ will not be Δ^0_2-categorical.

Let C be some Σ^0_2 set which is not Δ^0_2. By Lemma 3.13 there exists some $F : \mathbb{N} \times \mathbb{N} \to \{0, 1\}$ such that:

(i) $\forall x \ \{s : F(x, s) = 0\}$ is infinite

(ii) $x \in C \iff \{s : F(x, s) = 1\}$ is finite

Fix some such C and some such F. We will build g up in stages, g_s.

stage 0: We build one single 1-chain by defining g_0 as follows:

$g_0(1) \uparrow$

stage s-1: (inductive assumption) At the end of stage $s - 1$ we have the following:

- $\mathcal{O}_{g_{s-1}}\big((2i+1)2^{(i+1)n}\big)$ is defined $\forall i, n < s$
- For each $2i+1$ (with $i < s$), at most one orbit of $(2i+1)2^{(i+1)n}$ is marked. (Note however that that single orbit could contain $(2i+1)2^{(i+1)n}$ for many such n's.)
- g_{s-1} is defined as follows $\forall i, n < s$:

$$g_{s-1}\big((2i+1)2^{(i+1)n}\big) = (2i+1)2^{(i+1)n+1}$$
$$g_{s-1}\big((2i+1)2^{(i+1)n+1}\big) = (2i+1)2^{(i+1)n+2}$$
$$\vdots$$
$$g_{s-1}\big((2i+1)2^{(i+1)n+(i-1)}\big) = (2i+1)2^{(i+1)n+i}$$
$$g_{s-1}\big((2i+1)2^{(i+1)n+i}\big) \downarrow \iff F(i, n) = 1$$

stage s: We extend g_{s-1} as follows.

- For $i = s$, create $s+1$ finite chains of size $i+1$ by multiplying by 2 as needed. That is define g_s as follows for each $n \in \{0, 1, \ldots, s\}$:

$$g_s\big((2i+1)2^{(i+1)n}\big) = (2i+1)2^{(i+1)n+1}$$
$$g_s\big((2i+1)2^{(i+1)n+1}\big) = (2i+1)2^{(i+1)n+2}$$
$$\vdots$$
$$g_s\big((2i+1)2^{(i+1)n+(i-1)}\big) = (2i+1)2^{(i+1)n+i}$$
$$g_s\big((2i+1)2^{(i+1)n+i}\big) \uparrow$$

- For each $i < s$, create one additional finite chain of size $i+1$ by multiplying by 2. That is, for each $i < s$ and $n = s$ define the following:

$$g_s\big((2i+1)2^{(i+1)n}\big) = (2i+1)2^{(i+1)n+1}$$
$$g_s\big((2i+1)2^{(i+1)n+1}\big) = (2i+1)2^{(i+1)n+2}$$
$$\vdots$$
$$g_s\big((2i+1)2^{(i+1)n+(i-1)}\big) = (2i+1)2^{(i+1)n+i}$$
$$g_s\big((2i+1)2^{(i+1)n+i}\big) \uparrow$$

- For each $i, t \leq s$, calculate $F(i,t)$. If $F(i,t) = 0$ then do nothing. If $F(i,t) = 1$ then do the following:
  - Take the entire orbit of $(2i+1)2^{(i+1)t}$ and mark it. (Note that we are marking it as a potential ω^*-orbit.)
  - If no other orbits of $(2i+1)2^{(i+1)n}$ is marked for any n thus far (note that we have only built orbits to include up through $n = 0, 1, \ldots, s$ at this point), then do nothing.
  - If there exists an orbit such that $(2i+1)2^{(i+1)n}$ is marked for some $n \leq s$, then take the entire orbit of $(2i+1)2^{(i+1)t}$ and attach it to the beginning of the orbit for $(2i+1)2^{(i+1)n}$. That is, let a be the unique element of $\mathcal{O}_{g_s}\big((2i+1)2^{(i+1)n}\big)$ such that $a \notin \text{range}(g_s)$, and define $g_s\big((2i+1)2^{(i+1)t+i}\big) = a$. Leave the entire orbit marked.

    Note: this attaches $\mathcal{O}_{g_s}\big((2i+1)2^{(i+1)t}\big)$ to $\mathcal{O}_{g_s}\big((2i+1)2^{(i+1)n}\big)$, which gives us that $\mathcal{O}_{g_s}\big((2i+1)2^{(i+1)t}\big) = \mathcal{O}_{g_s}\big((2i+1)2^{(i+1)n}\big)$. Therefore at the end of this step of this stage we will have at most one orbit marked for each $2i+1$, even though this single orbit could contain $(2i+1)2^{(i+1)n}$ for several n's.

We let $g = \lim_s g_s$. In this way we get that $\mathcal{O}_g\big((2i+1)2^{(i+1)t}\big)$ is an ω^*-orbit $\iff$ $\{s : F(i,s) = 1\}$ is infinite and $F(i,t) = 1 \iff i \notin C$ and $F(i,t) = 1$.

We see that g is indeed a partial computable function since the above process for each g_s is a computable process at each stage s, and once $g_s(x) \downarrow = y$, then $\forall t > s$, $g_t(x) \downarrow = y$. Explicitly, given input x, to determine $g(x)$ we do the following.

- We first determine the "even" and "odd" parts of the prime decomposition of x. That is, we find i and k such that $x = (2i+1)2^k$.
- We then run the above process for as many stages as needed until $g_s((2i+1)2^k) \downarrow = y$.
- Once $g_s((2i+1)2^k) \downarrow = y$, then we have $g(x) \downarrow = y$.

Note that this process may compute forever searching for a stage, s, where $g_s(x) \downarrow$. This is ok, however, as we only need g to be partial computable.

Therefore $\mathcal{B} = (\mathbb{N} - \{0\}, g)$ is a partial computable injection structure. We now check that $\mathcal{B}$ has infinitely many ω^*-orbits and infinitely many finite chain orbits of every size. We do this by examining what happens for $F(i,s) = 0$ and $F(i,s) = 1$ when $i \in C$ and $i \notin C$, based on our construction and the results of Lemma 3.13:

- $i \in C \implies \{s : F(i,s) = 1\}$ is finite $\implies$ infinitely many s's for which $F(i,s) = 0$:
  If there are infinitely many s's for which $F(i,s) = 0$ then, by our construction, this gives us that there are infinitely many points at which we leave $(2i+1)2^{(i+1)s}$ alone and do <u>not</u> add its orbit to any other orbit. Since each $(2i+1)2^{(i+1)s}$ is created as a finite chain of size $i+1$ for each s, if we do not add its orbit to any other orbit, this means we leave it as a finite chain of size $i+1$ throughout the remainder of the construction. This gives us that: for each $i \in C$ there are infinitely many finite chains of size $i+1$.
- $i \in C \implies \{s : F(i,s) = 1\}$ is finite $\implies$ finitely many s's for which $F(i,s) = 1$:
  This gives us only finitely many stages during which we add the orbit of $(2i+1)2^{(i+1)s}$ to another orbit (namely the orbit for $(2i+1)2^{(i+1)n}$ for some n). This gives us one giant finite chain of size $\ell_i(i+1)$ (where ℓ_i simply = the number of s's for which $F(i,s) = 1$). Therefore: for each $i \in C$, there is one finite chain of size $\ell_i(i+1)$.
  Note: since we wish in the end for $\mathcal{B}$ to have infinitely many finite chains of <u>every</u> size, this will not adversely affect our construction.
- $i \notin C \implies \{s : F(i,s) = 1\}$ is infinite $\implies$ infinitely many s's for which $F(i,s) = 1$:
  Therefore there are infinitely many stages where we add the orbit of $(2i+1)2^{(i+1)s}$ to another orbit (namely the orbit for $(2i+1)2^{(i+1)n}$ for some n). Therefore: for each $i \notin C$ we get one ω^*-orbit, namely $\mathcal{O}_g\big((2i+1)2^{(i+1)s}\big)$ for some s such that $F(i,s) = 1$.

- $i \notin C \implies$ (by Lemma 3.13) $\{s : F(i,s) = 0\}$ is also infinite $\implies$ infinitely many s's for which $F(i,s) = 0$:

  This yields infinitely stages for which we leave $(2i+1)2^{(i+1)s}$ alone and do <u>not</u> add its orbit to any other orbit. Therefore: for each $i \notin C$, there are infinitely many finite chains of size $i+1$.

Taken together we have that for each $i \in C$ there are infinitely many finite chains of size $i+1$, and for each $i \notin C$ there are infinitely many finite chains of size $i+1$. Therefore $\forall i$ there are infinitely many finite chains of size $i+1$, and since the size of any finite chain must necessarily be ≥ 1, this yields that $\mathcal{B}$ therefore has infinitely many finite chains of every size. Additionally, for each $i \notin C$ we get a single ω^*-orbit. Since C is not computable, there are infinitely many $i \notin C$, and hence $\mathcal{B}$ has infinitely many ω^*-orbits. Therefore $\mathcal{B} \simeq \mathcal{A}$.

Let A_{ω^*} = the set of all elements in ω^*-orbits under f, and B_{ω^*} = the set of all elements in ω^*-orbits under g. By assumption, under $\mathcal{A}$, A_{ω^*} is computable. In $\mathcal{B}$, since $\mathcal{O}_g\big((2i+1)2^{(i+1)s}\big)$ is an ω^*-orbit $\iff i \notin C$, this gives the following:

$$x \in \omega^*\text{-orbit} \iff \text{for some } n \Big[\big(x = (2i+1)2^{(i+1)n}\big) \vee \big(x = (2i+1)2^{(i+1)n+1}\big) \\ \vee \cdots \vee \big(x = (2i+1)2^{(i+1)n+i}\big)\Big] \wedge \Big[F(i,n) = 1\Big] \wedge \Big[i \in \bar{C}\Big]$$

$$B_{\omega^*} = \{(2i+1)2^{(i+1)n+m} : n \in \mathbb{N} \ \wedge \ m \leq i \ \wedge \ F(i,n) = 1 \ \wedge \ i \in \bar{C}\} \\ \equiv_T C$$

Therefore, since C is a Σ^0_2 set which is not Δ^0_2, B_{ω^*} =the set of all elements of ω^*-orbits under g is a Π^0_2 set which is not Δ^0_2. Therefore there cannot be a Δ^0_2-isomorphism $h : \mathcal{A} \to \mathcal{B}$. If there were such an isomorphism, we would have a computable set, A_{ω^*}, taken via a Δ^0_2-function, h, which would yield a Δ^0_2 set B_{ω^*} – a contradiction. (Taken another way, if there were such an isomorphism, h, we could determine whether $x \in B_{\omega^*}$ given a $\mathbf{0}'$-oracle – simply ask whether $h^{-1}(x) \in A_{\omega^*}$– and hence B_{ω^*} would be a Δ^0_2-set, a contradiction). Therefore $\mathcal{B}$ cannot be Δ^0_2-isomorphic to $\mathcal{A}$, and hence $\mathcal{A}$ is not Δ^0_2-categorical.

□

Note that we can generalize the above case to infinitely many of any other type of infinite orbit – rather than adding finite chains to the beginning as we did for ω^*-orbits in the construction of the proof for Theorem 3.14, we simply add finite chains to the end (for ω-orbits) or to alternating both

sides (for Z-orbits). This yields the following corollaries.

Corollary 3.15. *Let $\mathcal{A} = (A, f)$ be a partial computable injection structure with infinitely many ω-orbits, infinitely many finite chains of every size, and no other types of orbits. Then $\mathcal{A}$ is not Δ^0_2-categorical.*

Corollary 3.16. *Let $\mathcal{A} = (A, f)$ be a partial computable injection structure with infinitely many Z-orbits, infinitely many finite chains of every size, and no other types of orbits. Then $\mathcal{A}$ is not Δ^0_2-categorical.*

We can also generalize the case given in Theorem 3.14 to situations where $\mathcal{A}$ has infinitely many finite chains with sizes of every multiple of k for some $k \in \mathbb{N}$. That is, for some $k \in \mathbb{N}$, $\mathcal{A}$ has infinitely many k-chains, infinitely many $2k$-chains, infinitely many $3k$-chains, etc. We simply modify the above construction so that instead of building finite chains in sizes of 1, 2, 3, etc, we build finite chains in sizes of $1k$, $2k$, $3k$, etc. If during the construction we stop and do not attach any more chains to a given orbit, this will not adversely affect the construction. We have only attached finite chains in multiples of k, so we are left with a finite chain with size of some multiple of k. Since there are infinitely many finite chains of size every multiple of k, additional ones will not adversely affect the structure.

This generalization serves as motivation for the following theorem, which says we can in fact generalize even one step further to consider infinitely many finite chains coming from any c.e. set K of arbitrarily large sizes.

Theorem 3.17. *Let $\mathcal{A} = (A, f)$ be a partial computable injection structure, and let K be some infinite c.e. set not containing 0. If $\mathcal{A}$ has infinitely many finite chain orbits of every size in K, and $\mathcal{A}$ has infinitely many infinite orbits (either infinitely many ω^*-, or infinitely many ω-, or infinitely many Z-orbits), and no other types of orbits, then $\mathcal{A}$ is not Δ^0_2 categorical.*

Proof. We first examine the case where $\mathcal{A}$ has infinitely many ω^*-orbits. We will later generalize this to the cases with infinitely many ω^*-orbits or infinitely many Z-orbits. Assume, then, that $\mathcal{A} = (\mathbb{N} - \{0\}, f)$, and that $\mathcal{A}$ has infinitely many ω^*-orbits, infinitely many finite chains of sizes coming from some infinite c.e. set K, and no other types of orbits. Furthermore, we assume that the set of all elements in ω^*-orbits is computable. (If not, we assume that our given structure is Δ^0_2-isomorphic to some such $\mathcal{A}$, and proceed. Note that if this is not the case, then we are done – our structure cannot be Δ^0_2-categorical.) Furthermore, we let K have enumeration $K = \{k_0, k_1, k_2, \ldots\}$, and since $0 \notin K$, then $0 \neq k_i$ $\forall i$.

Our goal will be to build some partial computable injection structure $\mathcal{B} = (\mathbb{N} - \{0\}, g)$ with infinitely many ω^*-orbits and infinitely many finite chains with sizes coming from $K = \{k_i\}_{i \geq 0}$, but in which the set of all elements in ω^*-orbits under g is not Δ^0_2. This will give us that $\mathcal{B} \simeq \mathcal{A}$, but $\mathcal{B}$ is not Δ^0_2-isomorphic to $\mathcal{A}$ (since if it were we would have a Δ^0_2-function applied to a computable set, yielding an isomorphic set that is not Δ^0_2). Hence $\mathcal{A}$ will not be Δ^0_2-categorical.

Let C be some Σ^0_2 set which is not Δ^0_2. By Lemma 3.13 there exists some $F : \mathbb{N} \times \mathbb{N} \to \{0, 1\}$ such that:

(i) $\forall x \ \{s : F(x, s) = 0\}$ is infinite

(ii) $x \in C \iff \{s : F(x, s) = 1\}$ is finite

Fix some such C and some such F. We will build g up in stages, g_s.

stage 0: We build one single k_0-chain by defining g_0 as follows for $i = 0$ and $n = 0$:

$$g_0\big((2i+1)2^{k_0 n}\big) = (2i+1)2^{k_0 n+1}$$

$$g_0\big((2i+1)2^{k_0 n+1}\big) = (2i+1)2^{k_0 n+2}$$

$$\vdots$$

$$g_0\big((2i+1)2^{k_0 n+(k_0-2)}\big) = (2i+1)2^{k_0 n+(k_0-1)}$$

$$g_0\big((2i+1)2^{k_0 n+(k_0-1)}\big) = \uparrow$$

In other words, we have:

$$g_0(1) = 2,\ g_0(2) = 4,\ \ldots,\ g_0(2^{k_0-2}) = 2^{k_0-1},\ g_0(2^{k_0-1}) = \uparrow$$

Let $j_{0,0} = 0$

stage s − 1: (inductive assumption) At the end of stage $s - 1$ we have the following:

- There are s-many finite chains of size k_i for all $i \leq s - 1$.
- Furthermore, we know which number k_i-chain each of the s many k_i-chains is, as we have kept track of it with $j_{i,t}$ so that $(2i+1)2^{j_{i,0}}$ marks the beginning of the 0th k_i-chain created during the construction, $(2i+1)2^{j_{i,1}}$ marks the beginning of the first k_i-chain created during the construction, and so on. $j_{i,t}$ has been defined for all $i, t \leq s - 1$.
- For each $(2i+1)2^j$, at most one orbit of $(2i+1)2^j$ is marked. (Note however that that single orbit could contain $(2i+1)2^j$ for many such j's.)
- For each $i \in \{0, 1, \ldots, s-1\}$ we have "specified" $g_{s-1}\big((2i+1)2^n\big)$ for only a finite number of n's thus far. Note that by "specified" we mean that this construction has specifically

stated whether $g_{s-1}\big((2i+1)2^n\big)\downarrow = y$ or whether $g_{s-1}\big((2i+1)2^n\big)\uparrow$. If we have said that $g_{s-1}\big((2i+1)2^n\big)\uparrow$, we still may make it halt at some later stage. Additionally, when we talk about some $g_{s-1}\big((2i+1)2^n\big)$ which has not been "specified", we mean that we haven't said anything about $g_{s-1}\big((2i+1)2^n\big)$ yet in the construction, not even possibly that $g_{s-1}\big((2i+1)2^n\big)\uparrow$.

stage s: We extend g_{s-1} as follows.

- For $i = s$, create $(s+1)$-many finite chains of size k_i by taking $(2i+1)$ and multiplying by 2 as needed. Specifically, for each $n \in \{0, 1, \ldots, s\}$, define g_s as follows:

  $$g_s\big((2i+1)2^{k_i n}\big) = (2i+1)2^{k_i n+1}$$
  $$g_s\big((2i+1)2^{k_i n+1}\big) = (2i+1)2^{k_i n+2}$$
  $$\vdots$$
  $$g_s\big((2i+1)2^{k_i n+(k_i-2)}\big) = (2i+1)2^{k_i n+(k_i-1)}$$
  $$g_s\big((2i+1)2^{k_i n+(k_i-1)}\big) = \uparrow$$

  Set $j_{i,n} = k_i n$ for $i = s$ and $n \in \{0, 1, \ldots, s\}$.

- For each $i < s$, create one additional finite chain of size k_i by taking the smallest such j for which $g_s\big((2i+1)2^j\big)$ has not yet been specified in the construction, and multiplying by 2 $(k_i - 1)$-many times:

  $$g_s\big((2i+1)2^{j}\big) = (2i+1)2^{j+1}$$
  $$g_s\big((2i+1)2^{j+1}\big) = (2i+1)2^{j+2}$$
  $$\vdots$$
  $$g_s\big((2i+1)2^{j+(k_i-2)}\big) = (2i+1)2^{j+(k_i-1)}$$
  $$g_s\big((2i+1)2^{j+(k_i-1)}\big) = \uparrow$$

  Set $j_{i,s} = j$.

- For each $i, t \leq s$, calculate $F(i,t)$. If $F(i,t) = 0$ then do nothing. If $F(i,t) = 1$ then do the following:
  - Take the entire orbit of $(2i+1)2^{j_{i,t}}$ and mark it. (Note that we are marking it as a potential ω^*-orbit.)
  - If no other orbits of $(2i+1)2^j$ are marked for any j specified by the construction thus far, then do nothing. This completes the stage for this $F(i,t) = 1$.
  - If there exists an orbit such that $(2i+1)2^j$ is marked for some j, then take the entire orbit of $(2i+1)2^{j_{i,t}}$ and attach it to the beginning of the already marked orbit

for $(2i+1)2^j$. That is, let a be the unique element of $\mathcal{O}_{g_s}\big((2i+1)2^j\big)$ such that $a \notin \text{range}(g_s)$, and define $g_s\big((2i+1)2^{j_{i,t}+k_i-1}\big) = a$. Leave the entire orbit marked. Note: at the end of this step of the stage, we will have at most one orbit marked for $(2i+1)^j$, even though this single orbit could contain many such j's.

- Now, for the single marked orbit for $(2i+1)^{j_{i,t}}$, determine its size. This is computably possible, as the marked orbit thus far was created by a finite number of computable steps, and the marked orbit is itself finite. Let $\text{card}\Big[\mathcal{O}_{g_s}\big((2i+1)2^{j_{i,t}}\big)\Big] = \mathscr{k}$
- Start searching through $K = \{k_i\}_{i\geq 0}$ until we find the smallest r such that $\mathscr{k} \leq k_r$. Note that K is infinite, meaning it contains arbitrarily large k_r's, meaning at some finite point we will indeed find k_r such that $\mathscr{k} \leq k_r$.
- If $\mathscr{k} = k_r$, then do nothing. This completes the stage for this $F(i,t) = 1$.
- If $\mathscr{k} < k_r$, then we will add $k_r - \mathscr{k}$ more elements from the 2^n multiples of $(2i+1)$ to make $\mathcal{O}_{g_s}\big((2i+1)2^{j_{i,t}}\big)$ have size k_r. Let j be the smallest number such that $g_s\big((2i+1)2^j$ has not yet been specified by our construction. Then let:

  $$g_s\big((2i+1)2^j\big) = (2i+1)2^{j_{i,t}}$$
  $$g_s\big((2i+1)2^{j+1}\big) = (2i+1)2^j$$
  $$\vdots$$
  $$g_s\big((2i+1)2^{j+k_r-\mathscr{k}-1}\big) = (2i+1)2^{j+k_r-\mathscr{k}-2}$$

  This gives that $\text{card}\Big[\mathcal{O}_{g_s}\big((2i+1)2^{j_{i,t}}\big)\Big] = k_r$

We let $g = \lim_s g_s$. In this way we get that $\mathcal{O}_g\big((2i+1)2^{j_{i,t}}\big)$ is an ω^*-orbit $\iff$ $\{s : F(i,s) = 1\}$ is infinite and $F(i,t) = 1 \iff i \notin C$ and $F(i,t) = 1$.

We see that g is indeed a partial computable function since the above process for each g_s is a computable process at each stage s, and once $g_s(x)\downarrow = y$, then $\forall t > s$, $g_t(x)\downarrow = y$. Explicitly, given input x, to determine $g(x)$ ...

- We first determine the "even" and "odd" parts of the prime decomposition of x. That is, we find i and j such that $x = (2i+1)2^j$.
- We then run the above process for as many stages as needed until $g_s((2i+1)2^j)\downarrow = y$.
- Once $g_s((2i+1)2^j)\downarrow = y$, then we have $g(x)\downarrow = y$.

Note that this process may compute forever searching for a stage, s, where $g_s(x)\downarrow$. This is ok, however, as we only need g to be partial computable.

Therefore $\mathcal{B} = (\mathbb{N} - \{0\}, g)$ is a partial computable injection structure. We now check that $\mathcal{B}$ has infinitely many ω^*-orbits and infinitely many finite chain orbits of every size in K. We do this by examining what happens for $F(i,s) = 0$ and $F(i,s) = 1$ when $i \in C$ and $i \notin C$, based on our construction and the results of Lemma 3.13:

- $\forall i$ (regardless of whether $i \in C$ or $i \notin C$), there are infinitely many s's for which $F(i,s) = 0$ because of the additional condition we added in Lemma 3.13. Therefore in our construction, there are infinitely s's for which we do nothing to the orbit of $(2i+1)2^{j_{i,s}}$. Since each $(2i+1)2^{j_{i,s}}$ was created as a finite chain of size k_i this gives us infinitely many finite chains of size k_i. Therefore for each i there are infinitely many finite chains of size k_i.

- $i \in C \implies \{s : F(i,s) = 1\}$ is finite $\implies$ finitely many s's for which $F(i,s) = 1$:
  This gives us only finitely many stages during which we add the orbit of $(2i+1)2^{j_{i,s}}$ to another orbit. Each time we do this, we ensure that we have a finite chain of size k_r for some $k_r \in K$. Therefore, on the final time that we add the orbit of $(2i+1)2^{j_{i,s}}$ to another orbit, we have created one additional finite chain of size k_r for some r. Therefore: for each $i \in C$, there is one additional finite chain of size k_r for some r.
  Note: since we already determined that there exist infinitely many finite chains of size k_i $\forall i$, this additional finite chain of size k_r changes nothing – there are still infinitely many finite chains of size k_i $\forall i$.

- $i \notin C \implies \{s : F(i,s) = 1\}$ is infinite $\implies$ infinitely many s's for which $F(i,s) = 1$:
  Therefore there are infinitely many stages where we add the orbit of $(2i+1)2^{j_{i,s}}$ to the orbit of $(2i+1)2^{j_{i,t}}$ for some $t < s$. Furthermore, when we do so, we always add the orbit of $(2i+1)2^{j_{i,s}}$ to the *beginning* of the other orbit. Therefore: for each $i \notin C$ we get one ω^*-orbit, namely $\mathcal{O}_g\big((2i+1)2^{j_{i,s}}\big)$ for some s such that $F(i,s) = 1$.

Taken together we have that $\mathcal{B}$ has infinitely many finite chains of size k_i for each i, hence infinitely many finite chains of every size in $K = \{k_i\}_{i \geq 0}$. Additionally, for each $i \notin C$ we get a single ω^*-orbit. Since C is not computable, there are infinitely many $i \notin C$, and hence $\mathcal{B}$ has infinitely many ω^*-orbits. Therefore $\mathcal{B} \simeq \mathcal{A}$.

Let A_{ω^*} = the set of all elements in ω^*-orbits under f, and B_{ω^*} = the set of all elements in ω^*-orbits under g. By assumption, under $\mathcal{A}$, A_{ω^*} is computable. In $\mathcal{B}$, since $\mathcal{O}_g\big((2i+1)2^{j_{i,s}}\big)$ is an

ω^*-orbit $\iff$ $i \notin C$, this gives the following:

$$\begin{aligned} B_{\omega^*} &= \{x : \mathcal{O}_g(x) \text{ is } \omega^*\text{-orbit}\} \\ &= \{(2i+1)2^j : i \notin C \ \wedge \ j, s \in \mathbb{N} \ \wedge \ \mathcal{O}_{g_s}\big((2i+1)2^j\big) \text{ is marked} \ \ \wedge \ F(i,s) = 1\} \\ &\equiv_T C \end{aligned}$$

Therefore, since C is a Σ^0_2 set which is not Δ^0_2, B_{ω^*} =the set of all elements of ω^*-orbits under g is a Π^0_2 set which is not Δ^0_2. Therefore there cannot be a Δ^0_2-isomorphism $h : \mathcal{A} \to \mathcal{B}$. If there were such an isomorphism, we would have a computable set, A_{ω^*}, taken via a Δ^0_2-function, h, which would yield a Δ^0_2 set B_{ω^*} – a contradiction. (Taken another way, if there were such an isomorphism, h, we could determine whether $x \in B_{\omega^*}$ given a $\mathbf{0}'$-oracle – simply ask whether $h^{-1}(x) \in A_{\omega^*}$– and hence B_{ω^*} would be a Δ^0_2-set, a contradiction). Therefore $\mathcal{B}$ cannot be Δ^0_2-isomorphic to $\mathcal{A}$, and hence $\mathcal{A}$ is not Δ^0_2-categorical.

In the case when $\mathcal{A}$ has infinitely many ω-orbits or infinitely many Z-orbits instead of infinitely many ω^*-orbits we can easily modify the above proof. For infinitely many ω-orbits, we assume that the set of elements in ω-orbits under $\mathcal{A}$ is computable and we grow the marked orbits in $\mathcal{B}$ by attaching to the end of the orbit, rather than to the beginning. For infinitely many Z-orbits, we assume that the set of elements in Z-orbits under $\mathcal{A}$ is computable and we grow the marked orbits in $\mathcal{B}$ by attaching to both sides – we can alternate stages when $F(i,t) = 1$, attaching first to one side and then to the other. □

3.3 Δ^0_3-Categoricity of Partial Injection Structures

Finally, we examine Δ^0_3-categoricity of partial injection structures. As before, we mention first what is already known about Δ^0_3-categoricity for injection structures.

Theorem 3.18 (Cenzer, Harizanov, and Remmel, [7])**.** *All computable injection structures are relatively Δ^0_3-categorical.*

Turning our attention now to partial computable injection structures, we get a corresponding result.

Theorem 3.19. *All partial computable injection structures are relatively Δ^0_3-categorical.*

Proof. Let $\mathcal{A} = (A, f)$ be a partial computable injection structure. The proof follows easily from Lemma 2.3, where we can see that all possible types of orbits in $\mathcal{A}$ have the needed classifications

within the arithmetical hierarchy. Specifically, we will show that $\mathcal{A}$ has a Σ^0_3 Scott family, consisting of Σ^0_3-computable formulas.

Let $\vec{d} = d_0, \ldots, d_r$ be a sequence in A. Then $\forall q \leq r$ we have that either d_q is part of a cycle, a chain, an ω-orbit, an ω^*-orbit, or a Z-orbit. This corresponds to one of the following:

(1) $\exists t\, f^k_t(d_q) = d_q$ (for some fixed, minimal $k > 0$)
$\iff \exists t \exists t_1 \ldots \exists t_{k-1} \Big[f^k(d_q) = d_q \wedge \big(f_{t_1}(d_q)\!\downarrow \neq d_q \wedge\ f^2_{t_2}(d_q)\!\downarrow \neq d_q \wedge \cdots \wedge f^{k-1}_{t_{k-1}}(d_q)\!\downarrow \neq d_q \big) \Big]$
(for some fixed $k > 0$)

(2) $\exists b \exists s \forall t \big[f^k_s(b) = d_q \wedge b \notin \text{range}(f) \wedge f^m_t(d_q) \uparrow \big]$ (for some fixed $k \geq 0$, $m > 0$ minimal)
$\iff \exists b \exists s \exists t_1 \ldots \exists t_{m-1} \forall a \forall t \Big[f^k_s(b) = d_q \wedge\ f_t(a) \neq b \wedge\ f^m_t(d_q) \uparrow$
$\wedge \big(f_{t_1}(d_q)\!\downarrow \neq d_q \wedge\ f^2_{t_2}(d_q)\!\downarrow \neq d_q \wedge \cdots \wedge f^{m-1}_{t_{m-1}}(d_q)\!\downarrow \neq d_q \big) \Big]$ (for some fixed $k \geq 0$, $m > 0$)

(3) $\Big[\exists b \exists s \big(f^k_s(b) = d_q \wedge b \notin \text{range}(f) \big) \Big] \wedge \Big[\forall j \exists t f^j_t(d_q)\!\downarrow \neq d_q \Big]$ (for some fixed $k \geq 0$)
$\iff \exists b \exists s \forall a \forall \sigma \forall j \exists t \big[f^k_s(b) = d_q \wedge f_\sigma(a) \neq b \wedge f^j_t(d_q)\!\downarrow \neq d_q \big]$ (for some fixed $k \geq 0$)

(4) $\forall j \exists b \exists t \left[f^j_t(b) = d_q \right] \wedge \forall s f^k_s(d_q) \uparrow$ (for some fixed, minimal $k > 0$)
$\iff \forall s \forall j \exists b \exists t \exists t_1 \ldots \exists t_{k-1} \Big[f^j_t(b) = d_q \wedge f^k_s(d_q) \uparrow \wedge \big(f_{t_1}(d_q)\!\downarrow \neq d_q \wedge\ f^2_{t_2}(d_q)\!\downarrow \neq d_q \wedge \cdots$
$\wedge f^{k-1}_{t_{k-1}}(d_q)\!\downarrow \neq d_q \big) \Big]$ (for some fixed $k > 0$)

(5) $\big(\forall j \exists b \exists t f^j_t(b) = d_q \big) \wedge \big(\forall m \exists s f^m_s(d_q)\!\downarrow \neq d_q \big)$
$\iff \forall j \forall m \exists b \exists t \exists s \big[f^j_t(b) = d_q \wedge f^m_s(d_q)\!\downarrow \neq d_q \big]$

Similar to our previous theorems, we note that in (1) if we have more than one element in our sequence $\vec{d}$ which meets this condition for the same value of k, then we have to specify whether they are in the same cycle or different cycles. Therefore for d_q, d_Q in our sequence $\vec{d}$ with $q \neq Q$ and $q, Q \leq r$, if both d_q and d_Q satisfy the same formula (1) with the same value of k then either:

(6) $\exists t\, f^j_t(d_q)\!\downarrow = d_Q$ (for some fixed $j < k$)

(7) $\forall j < k\ \exists t\, f^j_t(d_q)\!\downarrow \neq d_Q$

We also note that in (5) if we have more than one element in our sequence $\vec{d}$ which meets this condition, then we have to specify whether they are in the same Z-orbit or different Z-orbits. Therefore for d_q, d_Q in our sequence $\vec{d}$ with $q \neq Q$ and $q, Q \leq r$, if both d_q and d_Q satisfy the same formula (5) then either:

(8) $\exists t\ f^k_t(d_q) = d_Q$ (for some fixed $k \geq 0$)

(9) $\exists t\ f^k_t(d_Q) = d_q$ (for some fixed $k \geq 0$)

(10) $\forall k \forall t \; (f_t^k(d_q)\downarrow \neq d_Q \; \wedge \; f_t^k(d_Q)\downarrow \neq d_q)$

It is clear that (1) and (6)-(9) are Σ_1^0 statements, and hence are also Σ_3^0. It is also clear from the above characterization that (2) is a Σ_2^0 statement, (4) and (5) are Π_2^0 statements, and (10) is a Π_1^0 statement, hence all are also Σ_3^0 statements. Additionally, as written above, (3) is a Σ_3^0 statement. Each formula in our Scott family of formulas will look like some combination of (1)-(10) with d_q and d_Q replaced with x_q and x_Q respectively. This is clearly a countable family of Σ_3^0 formulas, as each of our fixed variables k and m range over $\mathbb{N}$. It is easy to see that each sequence $\vec{d}$ in A will have a formula in this family – we simply take the appropriate combination of (1)-(10). Finally, it is also clear that given two sequences in A which satisfy the same Scott formula in this family, then there exists an automorphism mapping one sequence to the other.

□

We observe that up until this point, our results about partial injection structures seemed to yield decidedly more "difficult" answers when compared to the corresponding results about injection structures. For computable categoricity, we had two classifications of relatively computably categorical structures, the second of which involved certainly more intricate conditions than the single classification for injection structures. In addition, for non-computable categoricity there were several more cases to examine, with more intricate details to keep track of. For Δ_2^0-categoricity, there was a nice corresponding classification for partial injection structures, but for the examination of non-Δ_2^0-categorical cases we had to break the results into more subcases, some of which relied upon having orbits from at least a c.e. set. At the level of Δ_3^0-categoricity, however, we see that our results for injection structures and partial injection structures match, as the outcomes in Theorem 3.18 and Theorem 3.19 are identical.

3.4 Index Sets of Partial Computable Injection Structures

We now wish to extend the concept of index sets to infinite partial computable injection structures, building off of work done in [7]. We focus on structures with universe $\mathbb{N}$, and we define $\mathcal{A}_e = (\mathbb{N}, \varphi_e)$, where φ_e is the eth partial computable function in our enumeration of all partial computable functions. Since φ_e lists all partial computable functions, the list of all $\mathcal{A}_e$'s certainly contains all partial computable injection structures with universe $\mathbb{N}$.

We wish to examine exactly when $\mathcal{A}_e$ corresponds to a partial computable injection structure, so we define the index set PInj= $\{e : \mathcal{A}_e$ is a partial computable injection structure $\}$. Clearly $\mathcal{A}_e$ is

a partial computable injection structure $\iff$ φ_e is an injection. We now note the simple fact that the property of being 1-1 is a Π^0_1-statement:

$$\varphi_e \text{ is 1-1} \iff \forall x \forall y \forall s \forall t \big[\varphi_{e,s}(x) \downarrow = \varphi_{e,t}(y) \implies x = y\big]$$

We take this one step further in the following theorem, which in fact gives Π^0_1-completeness.

Theorem 3.20. *The set PInj* $= \{e : \mathcal{A}_e$ *is a partial computable injection structure* $\}$ *is* Π^0_1*-complete.*

Proof. As noted above, PInj $= \{e : \varphi_e$ is 1-1 $\}$, and being 1-1 is a Π^0_1 property. It is therefore clear that PInj is a Π^0_1 set.

To prove that PInj is Π^0_1-complete we need then to show that any Π^0_1 set is m-reducible to it. We do this by showing that $\overline{K} \leq_m$ PInj, where $\overline{K}$ = the complement of the Halting set $= \{e : \varphi_e(e) \uparrow\}$. We note that due to Post's theorem, $\overline{K}$ is itself a Π^0_1-complete set, and hence we will have $\overline{K} \leq_m$ PInj $\implies \overline{K} \equiv_m$ PInj, and hence PInj will be Π^0_1-complete.

To show that $\overline{K} \leq_m$ PInj, we need to define an m-reduction, F, from $\overline{K}$ to PInj. That is, we wish to define a total computable function $F(x)$ such that $x \in \overline{K} \iff F(x) \in$ PInj $\iff \varphi_{F(x)}$ is 1-1. To do this, we define a function $f = \varphi_{F(x)}$ as follows.

$$f(x,s) = \varphi_{F(x)}(s) = \begin{cases} s+1 & \text{if } \varphi_{x,s}(x) \uparrow \\ 1 & \text{if } \varphi_{x,s}(x) \downarrow \end{cases}$$

Clearly, f is computable for any fixed x. To calculate $f(x,s)$, we simply calculate $\varphi_x(x)$ for s-many stages and see whether $\varphi_{x,s}(x)$ halts. Therefore we know by the s-m-n theorem that $F(x)$ is a total computable function.

We now have that $f = \varphi_F(x)$ defines a structure, $\mathcal{A}_{F(x)} = (\mathbb{N}, \varphi_{F(x)})$. We examine exactly when

$\mathcal{A}_{F(x)}$ is a partial computable injection structure.

$$
\begin{aligned}
e \in \overline{K} &\implies \varphi_e(e)\uparrow \\
&\implies \forall s\ \varphi_{e,s}(e)\uparrow \\
&\implies \forall s\ \varphi_{F(e)}(s) = s+1 \\
&\implies \varphi_{F(e)} \text{ is a 1-1 partial computable function} \\
&\implies \mathcal{A}_{F(e)} = (\mathbb{N}, \varphi_{F(e)}) \text{ is a partial computable injection structure} \\
&\implies F(e) \in \text{PInj} \\
e \in K &\implies \varphi_e(e)\downarrow \\
&\implies \exists s > 0\ \big(\varphi_(e,s)\downarrow \text{ and } \varphi_{e,s-1}\uparrow\big) \\
&\implies \varphi_{F(e)}(0) = 1 \text{ and } \varphi_{F(e)}(s) = 1 \\
&\implies \varphi_{F(e)} \text{ is not 1-1} \\
&\implies \mathcal{A}_{F(e)} = (\mathbb{N}, \varphi_{F(e)}) \text{ is not a partial computable injection structure} \\
&\implies F(e) \notin \text{PInj}
\end{aligned}
$$

Therefore, $F(\overline{K}) \subseteq \text{PInj}$ and $F(K) \subseteq \overline{\text{PInj}}$, and hence F yields the desired m-reduction. Therefore $\overline{K} \leq_m \text{PInj}$, completing the proof.

□

3.5 Future Research

We have now considered results about relative computable categoricity for partial injection structures in Theorems 2.4 and 2.5 and about relative Δ^0_2-categoricity in Theorems 3.2 and 3.9. We have also considered results about computable categoricity for partial injection structures in Theorems 2.7, 2.8 and 2.10 to 2.12, and about Δ^0_2-categoricity in Theorems 3.12, 3.14 and 3.17 and in Corollary 3.15 and Corollary 3.16. Our goal, of course, is to completely classify all types of partial computable injection structures in terms of computable categoricity and Δ^0_2-categoricity. The aforementioned theorems, taken together, have considered all combinations of orbits that could make up a partial injection structures, at least in some form.

Although we have covered the big-picture situations and concepts, we have not thoroughly considered every single last situation, however. Of obvious note is a corresponding result to Theorem 2.11

and Theorem 2.12, where our partial computable injection structure $\mathcal{A}$ has finite chains of arbitrarily large size or has infinitely many ℓ-chains and infinitely many "$\geq \ell$"-cycles, and furthermore those sizes of finite chains or cycles, $\{k_i\}_{i>0}$, cannot be enumerated in a computable manner. (Recall $\{k_i\}_{i>0}$ represents the sequence of all sizes of finite chains or of all sizes of "$\geq \ell$"-cycles, so that $\mathcal{A}$ has exactly one orbit of size k_i for each i.) The case where $\{k_i\}_{i>0}$ forms a sequence which is not even c.e. still needs to be researched, however. Furthermore, the investigation of this case will likely require methods other than those used here. All of our proofs have relied upon building a partial computable injection structure in stages based upon the sizes and types of orbits in the desired structure. If the arbitrarily large sized finite chains have sizes which are not even c.e., there seems to be no real feasible way to build such a structure up in stages. We will have to use different methods, then, to do this.

Similarly, we may wish to examine a corresponding result to Theorem 3.17 about Δ^0_2-categoricity. The case where our partial computable injection structure has infinitely many infinite orbits of at least one type and infinitely many finite chain orbits of every size in some non-c.e. set, K, requires investigation. Again, we will likely have to use different methods to examine this case.

An additional item of research is the following. If we look closely at the statements and proofs of Theorems 2.7, 2.8, 2.10 to 2.12, 3.12, 3.14 and 3.17, we see that we have discussed computable partial injection structures with only the exact type of specified orbits, but not ones which have both the specified types of orbits, and other types of orbits. For instance, though we have proven in Theorem 2.8 that a partial computable injection structure whose orbits consist of infinitely many k-chains and infinitely many m-chains is not computably categorical, we have not formally proven that a partial computable injection structure whose orbits consist of infinitely many k-chains, infinitely many m-chains, and some other types of orbits is not computably categorical. We hope, of course, that the proofs of these corresponding theorems are straightforward, as they should simply build upon the existing constructions. These proofs still require formalization, however, before we can definitively say that computable partial injection structures with the specified types of orbits, and additional types of orbits, are not computably categorical.

Chapter 4

Algorithmic Equivalence of Trees and Nested Equivalence Structures

In this chapter we build off of work done by Calvert, Cenzer, Harizanov, and Morozov in [6] and Cenzer, Harizanov, and Remmel in [10] in their study of equivalence structures, and off of work done by Lempp, McCoy, Miller, and Solomon in [31] and by Miller in [36] on the computable categoricity of trees of finite height and trees of infinite height. We will show a way to think of nested equivalence structures as trees, and we will formally present computable methods to go back and forth between the two. We begin first with a discussion of nested equivalence structures and trees.

4.1 Nested Equivalence Structures

An **equivalence structure**, $\mathcal{A}$, consists of a set, $A \subseteq \mathbb{N}$, and an equivalence relation, E, on A. That is, E is a relation on A which is reflexive, symmetric, and transitive. We generally write $\mathcal{A} = (A, E)$ to denote such a structure. $\mathcal{A}$ is said to be a **computable equivalence structure** if A is a computable set and E is a computable relation. Equivalence structures can be completely classified up to isomorphism by the sizes of their equivalence classes and the numbers of equivalence classes of each size.

We now extend the notion of equivalence structures to include more than one equivalence relation, each of which is nested together. Given two equivalence relations, E and R on a set A, we say that they are **nested** – or more specifically that E is nested inside of R – if each equivalence class under

E sits inside of the corresponding equivalence class under R. That is, for each $a \in A$,

$$[a]_E \subseteq [a]_R$$

We sometimes use the shorthand "$E \subseteq R$" to represent this nesting of the equivalence classes of each equivalence relation. Given two nested equivalence relations, $E \subseteq R$, we call R the **coarser** equivalence relation and E the **finer** equivalence relation, since the equivalence classes of R are "larger" (in the sense of subset inclusion) and result in theoretically a coarser division of A with "fewer" equivalence classes, and the equivalence classes of E are "smaller" (in the sense of subset inclusion) and result in theoretically a finer division of A with "more" equivalence classes. Indeed, both E and R could have infinitely many classes each of infinite cardinality, but we still reference them as coarser or finer in this way.

A **nested equivalence structure**, $\mathcal{A} = (A, E_1, \ldots, E_n)$, consists of a set, A, and equivalence relations $E_1, \ldots, E_n$ on A such that the equivalence classes of each E_{i+1} are nested inside of the equivalence classes for E_i. That is, for each $a \in A$:

$$[a]_{E_n} \subseteq [a]_{E_{n-1}} \subseteq \cdots \subseteq [a]_{E_2} \subseteq [a]_{E_1}$$

By convention, we list the relations in our structure $\mathcal{A}$ from coarsest (E_1), to finest (E_n). We sometimes call $\mathcal{A}$ an **n-nested equivalence structure** to emphasize that there are n different equivalence relations inside of the structure. We consider here only finitely nested equivalence structures. That is, we do not consider nested equivalence structures of the form $\mathcal{A} = (A, E_1, \ldots, E_n, \ldots)$. We say that $\mathcal{A}$ is a **computable nested equivalence structure** iff $A, E_1, \ldots, E_n$ are all computable. We can relativize this notion, so that given any nested equivalence structure, $\mathcal{A}$, we say that another structure, set, function, or relation is "$\mathcal{A}$-computable" if we have an oracle for $A \oplus E_1 \oplus E_2 \cdots \oplus E_n$.

A nice property of computable, finitely nested equivalence structures is that we can computably enumerate all of its equivalence classes without repetition. Let $C_{\mathcal{A}}$ = the set of equivalence classes of $\mathcal{A}$ without repetition. We can enumerate $C_{\mathcal{A}}$ as $\{C_{0,i_0}, C_{1,i_1}, \ldots\}$, where each member of $C_{\mathcal{A}}$ is a set of the form $C_{j,i_j} = [c_j]_{E_{i_j}}$, with $c_j \in A$, and $i_j \in \{1, \ldots, n\}$.

Lemma 4.1. *Let $\mathcal{A} = (A, E_1, \ldots, E_n)$ be a nested equivalence structure. Then $C_{\mathcal{A}}$ is an $\mathcal{A}$-computable set, and furthermore we can enumerate $C_{\mathcal{A}}$ without repetition.*

Proof. We must give some $\mathcal{A}$-computable process which will enumerate all equivalence classes of $\mathcal{A}$ exactly once each. First we note that clearly A is $\mathcal{A}$-computable, which means that given an

A-oracle, we can enumerate A as $A = \{a_0, a_1, a_2, \ldots\}$ where $a_0 < a_1 < a_2 < \cdots$. Fix such an enumeration of A. We also note that E_i is $\mathcal{A}$-computable for $i \in \{1, \ldots, n\}$. Below is such a process to enumerate $C_{\mathcal{A}}$ without repetition.

1. First, enumerate $[a_0]_{E_1}$, $[a_0]_{E_2}$, ..., $[a_0]_{E_n}$ into $C_{\mathcal{A}}$. (That is, $c_0 = a_0$, $c_1 = a_0$, ..., $c_{n-1} = a_0$, $i_0 = 1$, $i_1 = 2$, ..., i_{n-1}.)

2. Next, examine a_1 under E_i for each $i \in \{1, \ldots, n\}$ in order, starting with $i = 1$.

   - If $a_1 E_i a_0$, then do nothing. We have already enumerated the equivalence class $[a_1]_{E_i}$ (since $[a_1]_{E_i} = [a_0]E_i$).
   - If $\neg(a_1 E_i a_0)$ then enumerate the equivalence class $[a_1]_{E_1}$ into $C_{\mathcal{A}}$.

   (Note that determining whether two elements are equivalent under E_i is an $\mathcal{A}$-computable process for each i.)

3. Next, examine a_2 under E_i for each $i \in \{1, \ldots, n\}$ in order, starting with $i = 1$.

   - If $a_2 E_i a_0$ or $a_2 E_i a_1$, then do nothing. We have already enumerated the equivalence class $[a_2]_{E_i}$ (since $a_2 E_i a_0$ or $a_2 E_i a_1$).
   - If $\neg(a_2 E_i a_0)$ and $\neg(a_2 E_i a_1)$, then enumerate the equivalence class $[a_2]_{E_i}$ into $C_{\mathcal{A}}$.

   (Note again that determining whether two elements are equivalent under E_i is an $\mathcal{A}$-computable process for each i.)

4. Continue on in this manner, comparing each a_k to a_0, a_1, ..., a_{k-1} under each of E_1, ..., E_n in order. If at any point a_k is E_i-equivalent to one of its predecessors, we do nothing since this means we have already enumerated the E_i-equivalence class of a_k into $C_{\mathcal{A}}$. If, on the other hand, a_k is not E_i-equivalent to any of its predecessors, we enumerate $[a_k]_{E_i}$ into $C_{\mathcal{A}}$. Each of the comparisons made is $\mathcal{A}$-computable, and we conduct only a finite number of them for each a_k.

Therefore, each step of this process is $\mathcal{A}$-computable. Additionally, we only add new elements to $C_{\mathcal{A}}$ after having first checked that we did not already add them previously. In this manner, we therefore get an $\mathcal{A}$-computable enumeration of all equivalence classes of $\mathcal{A}$ without repetition.

Now, we have only thus far shown that $C_{\mathcal{A}}$ is $\mathcal{A}$-c.e. It is not necessarily obvious at this point that the above algorithm indeed yields an $\mathcal{A}$-computable set. To show that $C_{\mathcal{A}}$ is in fact $\mathcal{A}$-computable, we need to show that there exists an algorithm which can computably tell us whether a given

equivalence class is in $C_{\mathcal{A}}$. To do this, we first turn $C_{\mathcal{A}}$ into a set of natural numbers by choosing appropriate indices. We abuse notation slightly and identify the set $C_{\mathcal{A}}$ as defined above with the set of indices defining $C_{\mathcal{A}}$:

$$\begin{aligned} C_{\mathcal{A}} &= \{\langle j,i\rangle : [a_j]_{E_i} \in \text{ above algorithmic enumeration of } C_{\mathcal{A}}\} \\ &= \{\langle j,i\rangle : a_j \text{ is the smallest elt of } [a_j]_{E_i}\} \end{aligned}$$

(where a_j = the jth element in the $\mathcal{A}$-computable enumeration of A in order).

Now we are ready to give an algorithm which takes as input some $m \in \mathbb{N}$ and tells us definitively whether $m \in C_{\mathcal{A}}$. Below is such an algorithm:

1. Given $m \in \mathbb{N}$, determine $\langle j,i\rangle$ such that $\langle j,i\rangle = m$. (This is a computable process since encoding and decoding pairing functions is a computable process.)

2. Now, $\mathcal{A}$-computably determine the jth element of A in our fixed enumeration of A in $<$-order.

3. Then, compare a_j under E_i to all smaller elements of A. That is, ask the following: $a_jE_ia_0$? $a_jE_ia_1$? $\ldots$ $a_jE_ia_{j-1}$? (This is $\mathcal{A}$-computable since E_i is $\mathcal{A}$-computable for each $i \in \{1,\ldots,n\}$.)

   - If yes to at least one of these, that is, if $\big(a_jE_ia_0 \vee \ a_jE_ia_1 \vee \ \cdots \ \vee a_jE_ia_{j-1}\big)$, then stop. We know that $\langle j,i\rangle = m \notin C_{\mathcal{A}}$.
   - If no to all of these, that is, if $\big(\neg(a_jE_ia_0) \wedge \ \neg(a_jE_ia_1) \wedge \ \cdots \ \wedge \neg(a_jE_ia_{j-1})\big)$, then stop. We know that $\langle j,i\rangle = m \in C_{\mathcal{A}}$.

Now it is clear that $C_{\mathcal{A}}$ is an $\mathcal{A}$-computable set without repetition, in which each equivalence class of $\mathcal{A}$ is enumerated exactly once, and it is represented by the least such element in the equivalence class. That is,

$$[a_j]_{E_i} \in C_{\mathcal{A}} \iff a_j \text{ is the smallest elt of } [a_j]_{E_i} \tag{4.1}$$

□

A nice property of nested equivalence structures in general is an easy characterization of when two equivalence classes are contained in each other. We have the following.

Lemma 4.2. *Let $\mathcal{A} = (A, E_1, \ldots, E_n)$, let $a, b \in A$, and let $i, \ell \in \{1,\ldots,n\}$ such that $i \leq \ell$. Then,*

$$aE_ib \iff [b]_{E_\ell} \subseteq [a]_{E_i}$$

Proof. Since $\mathcal{A}$ is a nested equivalence structure with $i \leq \ell$, we know that $[b]_{E_\ell} \subseteq [b]_{E_i}$. Therefore, $aE_ib \implies [b]_{E_i} = [a]_{E_i} \implies [b]_{E_\ell} \subseteq [b]_{E_i} = [a]_{E_i}$. Conversely, we know that $b \in [b]_{E_\ell}$. Therefore $[b]_{E_\ell} \subseteq [a]_{E_i} \implies b \in [a]_{E_i} \implies bE_ia$. □

4.2 Trees

Trees have been a useful tool in the study of many areas of mathematics, and computable structure theory is no different. In examining computability-theoretic properties of nested equivalence structures, we build off of work done by Lempp, McCoy, Miller, and Solomon in [31] and by Miller in [36] on the computable categoricity of trees of finite height and trees of infinite height.

We define a **tree**, $\mathcal{T} = (T, \prec)$, to be a structure which consists of a universe, T, and a strict partial order, $\prec$, on T which obeys the following two conditions. First, T must contain a least element under $\prec$. And second, for every $x \in T$, $\prec$ well-orders the set of predecessors of x in T under $\prec$. For our purposes, we restrict ourselves to the case where T is a countable set. Note that there are several definitions of trees, of which this is just one. We have chosen this particular definition for several reasons, including that we can therefore follow the conventions used in [31] and [36], upon which much of this research is based. Additionally, this particular definition works much better with many of the properties of computability that we are interested in examining. For a slightly more detailed explanation of why this tree definition is preferred when examining computable categoricity, see [36].

We say that $\mathcal{T}$ is a **computable tree** if T is a computable set and $\prec$ is a computable relation. Each element of the universe T is called a **node** of the tree $\mathcal{T}$. We call the least element under $\prec$ the **root node**, or simply the root. We think of the root node as being at level 0 of the tree. Our convention here is that trees grow "down", with the root node being the top node of the tree. For a finite height tree, we define the **level of a node** $x \in T$ by counting the number of predecessors x has in T. That is:

$$\text{level}_{\mathcal{T}}(x) = \text{card}(\{y \in T : y \prec x\})$$

We define the **height of a tree** to be the largest level of the nodes in the tree. Note that this definition puts both the height and level of the root node at 0, and allows for possibly infinite height trees.

$$\text{ht}(\mathcal{T}) = \sup_{x \in T}\{\text{level}_{\mathcal{T}}(x)\}$$

Informally, we can think of a path through a tree, T, as being a list of nodes which begins at

the "top" of the tree, and goes all the way through to the "bottom" of the tree. Formally, a **path** through a tree $\mathcal{T} = (T, \prec)$ is a maximal linearly ordered subset of T. (Recall that a maximal linearly ordered subset is one that is not contained in any other linearly ordered subset of T.)

We say that a tree of finite height $n < \omega$ is a **full tree** if every node of the tree terminates at exactly level n. With these definitions, a full finite height tree of height n will have that all paths through the tree are paths containing $n + 1$ elements, or more simply are paths of length $n + 1$. (Note that other conventions exist for these definitions — ones where the height and length of such a path are defined to be equal, putting the root node at level 0 and height 1. We have chosen these specific conventions for ease of notation later on.)

We now show some nice computability-theoretic properties when dealing with full computable trees of finite height. Although the following results are pretty standard in the study of trees, they will be quite useful in later sections so we discuss them in detail here.

It is important to note that in the general setting, the level of a node of a tree is not a computable function, though for a computable tree it is c.e. since we can approximate it in stages. If we restrict our view to only full finite height trees, however, this changes.

Lemma 4.3. *If $\mathcal{T}$ is a full computable tree of finite height, n, then determining the level of a given node is a computable process. (That is, $\text{level}_{\mathcal{T}}(x)$ is a computable function.)*

Proof. Let $\mathcal{T} = (T, \prec)$ be a full computable tree of finite height, n, and let x be some node in $\mathcal{T}$. Since $\mathcal{T}$ is full of height n, all paths through $\mathcal{T}$ are exactly length $n+1 = \text{ht}(\mathcal{T})+1$, and this means that all nodes in $\mathcal{T}$ are in a path of length $n + 1$. Hence, in any algorithmic process to determine $\text{level}_{\mathcal{T}}(x)$, we know exactly when to stop waiting! Explicitly, given some node x of our tree $\mathcal{T}$, we describe such a computable process to determine $\text{level}_{\mathcal{T}}(x)$:

1. Start enumerating elements of T. (Possible computably since universe of T is a computable set.)

2. As we enumerate elements of T, start pairwise determining whether pairs $y, z \in T$ are: $y \prec z$, $z \prec y$, or neither. (This is possible computably, since "$\prec$" is computable and at each stage we have enumerated only a finite list of elements into the universe of $\mathcal{T}$, hence at each stage there are only finitely many pairs of elements for which to determine "$\prec$".)

3. Continue this method until a stage s when we have enumerated x into T in step 1, and x is part of a chain under $\prec$ with exactly $n + 1$ elements in it from step 2. (We know such a finite

stage exists, since we know x must be in at least one path through $\mathcal{T}$, and every path is exactly length $n+1$.)

4. Count the number of predecessors of x in the chain. This will be the level of node x.

□

Note that in the above proof, we used the fact that T and $\prec$ were computable. We can in fact loosen that criteria, to get the following corollary. We use the shorthand notation that for a tree, $\mathcal{T} = (T, \prec)$, having a "$\mathcal{T}$-oracle" means that we have an oracle for $T \oplus \prec$.

Corollary 4.4. *If $\mathcal{T}$ is a full tree of finite height n, then $\text{level}_{\mathcal{T}}(x)$ is $\mathcal{T}$-computable.*

Proof. We modify the algorithm given in Lemma 4.3 as follows. Everywhere we "enumerate elements of T" or "determine whether pairs are $\prec$", we instead "use a $(T \oplus \prec)$-oracle to enumerate elements of T" and "use a $(T \oplus \prec)$-oracle to determine whether pairs are $\prec$". The rest follows. □

For computable, full finite height trees, we have a similar result for the set of predecessors of a given node. Let x be some node in a tree $\mathcal{T}$, and define $P_x = \{\text{predecessors of } x\} = \{y \in T : y \prec x\}$. The next result says that P_x is a computable set.

Lemma 4.5. *If $\mathcal{T} = (T, \prec)$ is a full computable tree of finite height, n, then determining the set of all predecessors of a given node is a computable process, uniform in n.*

Proof. Let $\mathcal{T} = (T, \prec)$ be a full computable tree of finite height n, and let x be some node in $\mathcal{T}$. Note that by definition we know that $\text{level}_{\mathcal{T}}(x) \leq n$. We can compute P_x as follows.

1. Compute $\text{level}_{\mathcal{T}}(x)$. (This is computable by Lemma 4.3.) Say $\text{level}_{\mathcal{T}}(x) = \ell$.

2. Start enumerating elements of T. As we enumerate each new element, compare it to x under $\prec$. (This is possible computably since T and $\prec$ are both computable.)

3. Keep enumerating elements of T until we have found ℓ-many elements which are $\prec x$. (We know this is a finite process since $\ell = \text{level}_{\mathcal{T}}(x) = \text{card}(\{y \in \mathcal{T} : y \prec x\})$.)

4. Let $P_x =$ the set of all ℓ-many nodes which are $\prec x$ from step 3.

□

Again, we can loosen the criteria that $\mathcal{T}$ be computable and get the following corollary.

Corollary 4.6. *If $\mathcal{T}$ is a full finite height tree of height n, then $P_x = \{\text{predecessors of } x\} = \{y \in T : y \prec x\}$ is $\mathcal{T}$-computable, uniformly in n.*

Proof. We modify the algorithm given in Lemma 4.5 to use a T-oracle and $\prec$-oracle in enumerating elements of T, determining whether elements are in T, or determining whether one element is $\prec$ another. The rest follows. □

Note that for all of the above concepts, we relied heavily on the fact that we knew how long each path in the tree was. Lempp, McCoy, Miller, and Solomon in [31] referred to this as a node being *established* at some particular stage. Specifically, if $\mathcal{T}$ is a computable tree of height n and $\{\mathcal{T}_s\}_{s>0}$ is some approximation of the tree, then a node x is **established at stage** s if it lies on a path of length $n+1$ and that path is entirely contained within the approximation $\mathcal{T}_s$. Though any node in a finite height computable tree may be established, the key concept behind full finite height computable trees is that *all* nodes of the tree are indeed established at some finite stage. We will take full advantage of this fact in later sections.

We now examine the different computable (or $\mathcal{T}$-computable) ways in which we can enumerate the nodes of $\mathcal{T}$. Let $T_i = \{\text{nodes at level } i\}$.

Lemma 4.7. *If $\mathcal{T} = (T, \prec)$ is a tree, then the set of nodes T is $\mathcal{T}$-computable, and furthermore we can $\mathcal{T}$-computably list the nodes in order $T = \{t_0, t_1, t_2, \cdots\}$ where $t_0 < t_1 < t_2 < \cdots$.*

Proof. The proof is straightforward, since being $\mathcal{T}$-computable means we have an oracle for T. Beginning with 0, we simply ask for each $n \in \mathbb{N}$ whether $n \in T$. The 0th n for which this is true is t_0, the m-th n for which this is true is t_m, and so on. This is clearly a $\mathcal{T}$-computable process. □

Lemma 4.8. *If $\mathcal{T} = (T, \prec)$ is a full tree of height n, then $T_i = \{\text{nodes at level } i\}$ is $\mathcal{T}$-computable for each $i \in \{0, \ldots, n\}$. Furthermore, we can list each of the elements in each T_i in order from smallest to largest under the usual ordering "$<$".*

Proof. We describe a $\mathcal{T}$-computable process to enumerate each T_i in order. First as per Lemma 4.7 we $\mathcal{T}$-computably enumerate the nodes of $\mathcal{T}$ in order: $T = \{t_0, t_1, t_2, \ldots\}$ with $t_0 < t_1 < t_2 < \cdots$. For each t_j, as we enumerate it we then determine $\text{level}_{\mathcal{T}}(t_j)$; this is $\mathcal{T}$-computable by Lemma 4.3. Then, we put each t_j into T_i accordingly where $i = \text{level}_{\mathcal{T}}(t_j)$. In this way, we let $m_{i,k}$ be the kth node at level i that is enumerated into T. Then $T_i = \{m_{i,0}, m_{i,1}, m_{i,2}, \ldots\}$ for each $i \in \{0, 1, \ldots, n\}$. Since each t_j is enumerated in order, so too is each $m_{i,k}$, and we have $m_{i,0} < m_{i,1} < m_{i,2} < \cdots$ for each $i \in \{0, 1, \ldots, n\}$, as desired. □

We have proved this for all levels of a full finite tree, $\mathcal{T}$. In particular, we will later need the above result for terminating nodes, or end nodes, of $\mathcal{T}$. If $\mathcal{T}$ is a full tree of height n, we will denote the set of all terminating nodes or end nodes of $\mathcal{T}$ enumerated in this way as $T_n = \{e_0, e_1, e_2, \ldots\}$. If we are perhaps dealing with two trees, say $\mathcal{T}$ and $\mathcal{S}$, then to clarify we will call their end nodes $\{e_{\mathcal{T},0}, e_{\mathcal{T},1}, e_{\mathcal{T},2}, \ldots\}$ and $\{e_{\mathcal{S},0}, e_{\mathcal{S},1}, e_{\mathcal{S},2}, \ldots\}$ respectively.

4.3 Basic Notions: Drawing a Tree from a Nested Equivalence Structure

We are now ready to begin examining nested equivalence structures further. We begin first with an intuitive discussion of how to represent a finitely nested equivalence structure as a finite-height tree. Doing so will not only visually help us view the nested equivalence structure and its associated (possibly quite intricate!) nesting properties, but it will also serve as a foundation for formally building a way to go back and forth between trees and nested equivalence structures, which we will do in Section 4.5 and Section 4.4.

Our goal, then, for the moment is: given a nested equivalence structure, $\mathcal{A} = (A, E_1, \ldots, E_n)$, we wish to represent it on a tree. We will do this by letting each node on the tree represent a unique equivalence class under one of the equivalence relations. The nodes at level i will represent the equivalence classes under E_i, and the branching of the tree will represent the subset inclusion, "$\subseteq$" of each equivalence class.

Method for drawing a nested equivalence structure tree:

1. Draw one root node at level 0.

2. Emanating from the root node, draw one branch for every equivalence class of E_1, and label each node with the names of the E_1 equivalence classes.

3. The first node at level 1 should now be labelled as $[a]_{E_1}$ for some $a \in A$. Emanating from this node, draw one branch for every equivalence class of E_2 which is a subset of $[a]_{E_1}$, and label each with the names of these E_2-equivalence classes.

4. Do the same for the 2nd and subsequent nodes at level 1, drawing one branch for every equivalence class of E_2 which is a subset of the node it emanates from, and labeling them with the appropriate E_2 equivalence classes.

5. Continue in this manner for each level of the tree until you have drawn branches and nodes to represent each of the equivalence classes for $E_3, \ldots, E_n$ at levels 3 through n of the tree.

6. Finally, for level $n+1$ of the tree, draw one branch emanating from each E_n-equivalence class for every element of A which is in the given E_n-equivalence class.

The result will be a tree which represents the nested equivalence structure by fully describing the nesting of the equivalence relations and the elements of each equivalence class. To more fully understand what is going on, this method is much better explained by an example.

Example 4.9. Draw the tree associated with the nested equivalence structure $\mathcal{A} = (\mathbb{N}, E_1, E_2, E_3)$, with E_1, E_2, and E_3 defined as follows $\forall n, m \in \mathbb{N}$:

$\underline{E_1}$	$\underline{E_2}$	$\underline{E_3}$
$2nE_12m$	$4nE_24m$	$8nE_3(8n+4)$
$(2n+1)E_1(2m+1)$	$(4n+1)E_2(4m+1)$	$(8n+1)E_3(8n+5)$
	$(4n+2)E_2(4m+2)$	$(8n+2)E_3(8n+6)$
	$(4n+3)E_2(4m+3)$	$(8n+3)E_3(8n+7)$

Solution. Notice that under each of E_1, E_2, E_3 we have the following equivalence classes:

E_1 : $[0]_{E_1} = \{0, 2, 4, 6, 8, 10, 12, \ldots\}$, $[1]_{E_1} = \{1, 3, 5, 7, 9, 11, 13, \ldots\}$

E_2 : $[0]_{E_2} = \{0, 4, 8, 12, \ldots\}$, $[1]_{E_2} = \{1, 5, 9, 13, \ldots\}$, $[2]_{E_2} = \{2, 6, 10, \ldots\}$, $[3]_{E_2} = \{3, 7, 11, \ldots\}$

E_3 : $[0]_{E_3} = \{0, 4\}$, $[1]_{E_3} = \{1, 5\}$, $[2]_{E_3} = \{2, 6\}$, $[3]_{E_3} = \{3, 7\}$,

$[8]_{E_3} = \{8, 12\}$, $[9]_{E_3} = \{9, 13\}$, $[10]_{E_3} = \{10, 14\}$, $[11]_{E_3} = \{11, 15\}$, $\ldots$

This yields the following nesting:

$$
\begin{aligned}
E_2 \subseteq E_1: \quad & [0]_{E_2}, [2]_{E_2} \subseteq [0]_{E_1} \\
& [1]_{E_2}, [3]_{E_2} \subseteq [1]_{E_1} \\
E_3 \subseteq E_2: \quad & [0]_{E_3}, [8]_{E_3}, [16]_{E_3}, \ldots \subseteq [0]_{E_2} \\
& [1]_{E_3}, [9]_{E_3}, [17]_{E_3}, \ldots \subseteq [1]_{E_2} \\
& [2]_{E_3}, [10]_{E_3}, [18]_{E_3}, \ldots \subseteq [2]_{E_2} \\
& [3]_{E_3}, [11]_{E_3}, [19]_{E_3}, \ldots \subseteq [3]_{E_2}
\end{aligned}
$$

Our tree then looks as follows:

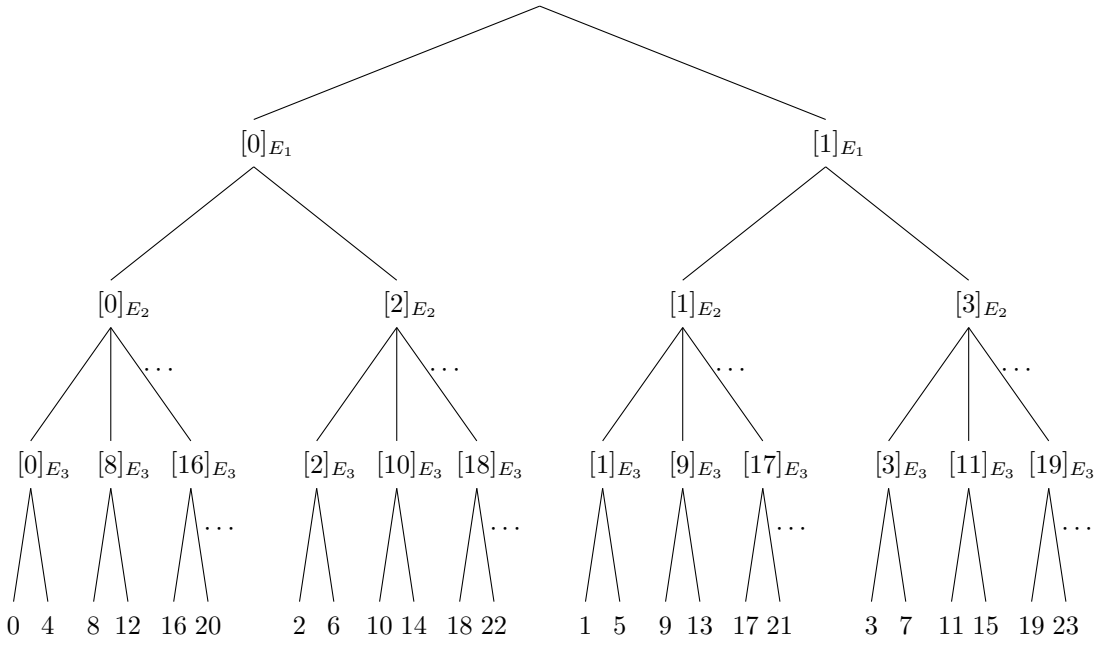


Figure 4.1: Tree representing Example 4.9, a 3-nested equivalence structure with its initial labelling of nodes.

□

Returning now to the general case there remains only one final step in labelling the tree we have now built for a nested equivalence structure of the form $\mathcal{A} = (A, E_1, \ldots, E_n)$. For consistency on each level, it is often nice to think of the root node and the terminating nodes as also representing equivalence classes. Let E_0 represent the equivalence relation in which everything is equivalent — that is, $\forall a, b \in A \left(aE_0b\right)$. Let E_{n+1} represent the equivalence relation of equality — that is, no elements of A are considered equivalent under E_{n+1}, unless they are identical. Note that these new

equivalence relations nest nicely inside of any nested equivalence structure as we would expect:

$$E_{n+1} \subseteq E_n \subseteq E_{n-1} \subseteq \cdots \subseteq E_2 \subseteq E_1 \subseteq E_0$$

Our final steps in creating the tree are to:

7. Label the root node as the single equivalence class for E_0.

8. Take each terminating node which is currently labeled "a" for some $a \in A$, and relabel it as $[a]_{E_{n+1}}$. The terminating nodes now correspond to E_{n+1} equivalence classes.

Performing these final steps on Example 4.9, yields the following labelling of our tree:

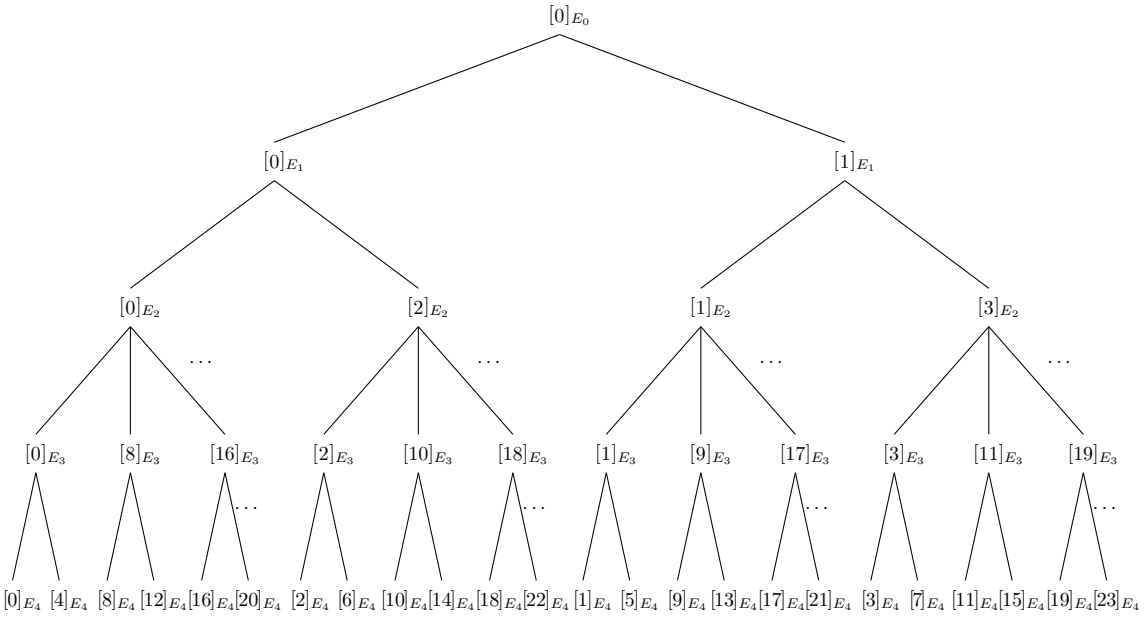


Figure 4.2: Tree representing Example 4.9, a 3-nested equivalence structure with all of its nodes labelled.

This completes the construction. Of course, these steps yield only an intuitive definition of a tree. To formally define a tree, we need to define the universe, and the partial order, and show that it obeys our desired properties. We will do this fully in the following section.

One thing to notice about this construction is that it appears to be a computable construction; the algorithm we have described seems like it could be applied to any computable nested equivalence structure. We will formally prove that this is indeed the case in the next section. The next thing to notice is the tree that we have built is a full tree. That is, all nodes terminate at the same level of the tree. The next thing to notice is that we started out with an n-nested equivalence structure

with $n = 3$, and the tree we built was of height $n + 1 = 4$. So a 1-nested equivalence structure would yield a full tree of height 2. It is then easy to intuitively see that we could take any full tree of height ≥ 2 and conversely interpret it as a nested equivalence structure.

4.4 Trees to Nested Equivalence Structures

In this section we formalize the notion of going from a tree to a nested equivalence structure. Although the methods we will describe would work equally as well for trees and nested equivalence structures with finite domains, we wish to focus our examination on trees and nested equivalence structures with countably infinite domains.

Given some full tree $\mathcal{T}$ of finite height ≥ 2 we wish to build a nested equivalence structure out of it. We will do this in several steps. First, we will create a computably isomorphic tree by simply relabeling the nodes in a computable manner. We will create the new labels of the tree in such a way that they are the pre-cursor to the equivalence classes, and we will call this isomorphism h. Then we will take the newly relabelled nodes, and use the order relation inherited from $\mathcal{T}$ to define a nested equivalence structure. We will call this second function $\tilde{h}$. Below is a picture explaining just what we will do.

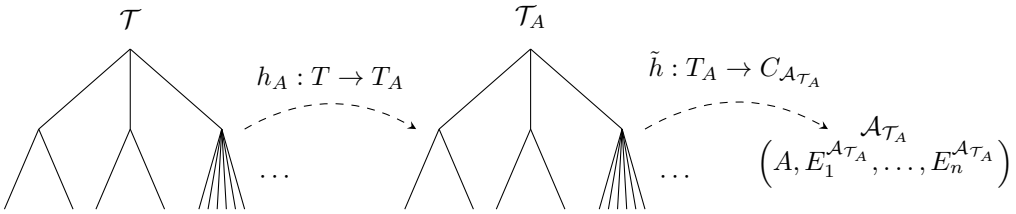


Figure 4.3: Diagram representing the process to take a full finite height tree, $\mathcal{T}$, with countably infinite universe and a countably infinite set, A, and build a nested equivalence structure from it.

In all of the steps to go between trees and nested equivalence structures we will prove that h and $\tilde{h}$ are 1-1, onto, computable, and preserve or create the desired structure. We first begin with h in the following theorem. Note that since our eventual goal is a nested equivalence structure of the form $\mathcal{A} = (A, E_1^{\mathcal{A}_{\mathcal{T}_A}}, \ldots, E_n^{\mathcal{A}_{\mathcal{T}_A}})$, it will be convenient to have the height of the given tree represented as $n + 1$. Additionally, we use the shorthand notation $(\mathcal{T} \oplus A)$ to represent the set $(T \oplus \prec_T \oplus A)$.

Theorem 4.10. *Given a tree $\mathcal{T} = (T, \prec_{\mathcal{T}})$ which is full and has finite height $n + 1 \geq 2$, and given any infinite set $A \subseteq \mathbb{N}$, we can define a function $h_A^{\mathcal{T}}$ which will $(\mathcal{T} \oplus A)$-computably build an isomorphic tree, $\mathcal{T}_A = (T_A, \prec_{\mathcal{T}_A})$, in which each node is of the form $[a]_{E_i^{\mathcal{A}_{\mathcal{T}_A}}}$ for some $a \in A$ and*

some $i \in \{0, \ldots, n+1\}$, and in which the tree $\mathcal{T}_A$ is itself $(\mathcal{T} \oplus A)$-computable.

Proof. The proof is contained in the following lemmas: Lemma 4.11, Lemma 4.12, and Lemma 4.13. □

We note that our function $h_A^{\mathcal{T}}$, which we have yet to define, depends upon the given set A and upon the tree $\mathcal{T}$. Given any set $A \subseteq \mathbb{N}$ and any tree $\mathcal{T}$ with infinite domain in $\mathbb{N}$, we will prove that we can define a function $h_A^{\mathcal{T}}$ via the below methods. When the tree $\mathcal{T}$ is clear from the context (as it is in the remainder of this chapter), we will drop the superscript $\mathcal{T}$, and write simply h_A. In later chapters (especially Chapter 5), the tree upon which $h_A^{\mathcal{T}}$ depends is often not clear from context, and so we will again use the subscript "$\mathcal{T}$" (or "$\mathcal{S}$", or "$\mathcal{U}$", etc) as necessary.

Now to define h_A, we let T be some infinite set, and we let $\mathcal{T} = (T, \prec_{\mathcal{T}})$ be a full tree of finite height $n+1 \geq 2$. Additionally, let $A = \{a_0, a_1, \ldots\} \subseteq \mathbb{N}$ with $a_0 < a_1 < \cdots$, and let $\{e_0, e_1, \ldots\}$ be an enumeration of all the end nodes of $\mathcal{T}$ as in Lemma 4.8. For $x \in T$, we let j represent the smallest i for which $x \preceq_{\mathcal{T}} e_i$. We can now define the function h_A as follows.

$$h_A(x) \stackrel{\text{defn}}{=} [a_j]_{E^{\mathcal{A}_{\mathcal{T}_A}}_{\text{level}_{\mathcal{T}}(x)}} \tag{4.2}$$

Note that at this point, we have a purely syntactical definition of h_A. We do not yet know what "$E^{\mathcal{A}_{\mathcal{T}_A}}$" means. It is simply notation at this point, to which we will later assign meaning. To calculate $h_A(x)$, we simply substitute a_j and $\text{level}_{\mathcal{T}}(x)$ into the above equation.

Lemma 4.11. *The function h_A, defined in Equation 4.2, is $(\mathcal{T} \oplus A)$-computable.*

Proof. We first A-computably fix some enumeration $A = \{a_0, a_1, a_2, \ldots\}$ such that $a_0 < a_1 < a_2 < \cdots$. We next fix some $\mathcal{T}$-computable enumeration of terminating nodes, or end nodes, of $\mathcal{T}$ as in Lemma 4.8: $T_{n+1} = \{e_0, e_1, e_2, \ldots\}$ with $e_0 < e_1 < e_2 < \cdots$. For the end nodes of $\mathcal{T}$ we define h_A to be:

$$h_A(e_j) = [a_j]_{E^{\mathcal{A}_{\mathcal{T}_A}}_{n+1}}$$

We now need to define h_A for inputs on other levels of $\mathcal{T}$.

1. Start with e_0. Now $\mathcal{T}$-computably find all predecessors of e_0 and their levels (as in Lemma 4.5 and Lemma 4.3). Order them by their levels as: $P_{e_0} = \{p_{e_0,0}, p_{e_0,1}, p_{e_0,2}, \ldots, p_{e_0,n}\}$, where the second index denotes the level of the given predecessor of e_0. Define h_A to be:

$$h_A(p_{e_0,i}) = [a_0]_{E_i^{\mathcal{A}_{\mathcal{T}_A}}}$$

2. Next, continue on to e_1. Now $\mathcal{T}$-computably find its predecessors and levels (as in Lemma 4.5 and Lemma 4.3), ordered as $P_{e_1} = \{p_{e_1,0}, p_{e_1,1}, p_{e_1,2}, \ldots, p_{e_1,n}\}$. Next check for $i \in \{0, \ldots, n\}$ whether $p_{e_1,i} = p_{e_0,i}$. (Note: at a minimum, $p_{e_1,0} = p_{e_0,0}$ since there is only one root node on $\mathcal{T}$.) Then, for each $i \in \{0, \ldots, n\}$:

   - If $p_{e_1,i} = p_{e_0,i}$, then do nothing. We have already defined h_A on this node.
   - If $p_{e_1,i} \neq p_{e_0,i}$, then define h_A to be:

$$h_A(p_{e_1,i}) = [a_1]_{E_i^{\mathcal{A}\mathcal{T}_A}}$$

3. Continue on in this manner, $\mathcal{T}$-computably enumerating predecessors of each e_k in order as $P_{e_k} = \{p_{e_k,0}, p_{e_k,1}, p_{e_k,2}, \ldots, p_{e_k,n}\}$, and then checking for which previously enumerated predecessors $p_{e_k,i} = p_{e_j,i}$.

   - If for some $j < k$ we have that $p_{e_k,i} = p_{e_j,i}$ then do nothing. We have already defined h_A on this node.
   - If $\forall j < k$ we have $p_{e_k,i} \neq p_{e_j,i}$ then define h_A to be:

$$h_A(p_{e_k,i}) = [a_k]_{E_i^{\mathcal{A}\mathcal{T}_A}}$$

The above is a $(\mathcal{T} \oplus A)$-computable process, and hence h_A is a $(\mathcal{T} \oplus A)$-computable function. Given some node $x \in T$, to figure out $h_A(x)$ we simply apply the above process in "reverse". That is, we $\mathcal{T}$-computably ask in this order: $x \preceq_{\mathcal{T}} e_0$? $x \preceq_{\mathcal{T}} e_1$? $x \preceq_{\mathcal{T}} e_2$? ... We know that there exists at least one i for which $x \preceq_{\mathcal{T}} e_i$ since x is a node on our full tree and $\{e_0, e_1, \ldots\}$ lists all end nodes of the tree. Then for $j =$ the smallest i for which $x \preceq_{\mathcal{T}} e_i$, we have that $h_A = [a_j]_{E^{\mathcal{A}\mathcal{T}_A}_{\text{level}_{\mathcal{T}}(x)}}$. □

Now, although we know that our end-goal is to use these relabelled nodes of the tree to eventually represent equivalence classes of a nested equivalence structure, at this point we have purely a syntactic definition of $[a_j]_{E_i^{\mathcal{A}\mathcal{T}_A}}$ based on our algorithm for h_A. So although $[a_j]_{E_i^{\mathcal{A}\mathcal{T}_A}}$ looks like an equivalence class, we have yet to define the equivalence relation or any other members of the class. At this point in our process, then, $[a_j]_{E_i^{\mathcal{A}\mathcal{T}_A}}$ is purely some syntactical combination of symbols as applied by the above algorithm. There is no deeper meaning (yet!) implied, and therefore for the moment:

$$\left([a_j]_{E_i^{\mathcal{A}\mathcal{T}_A}} = [a_k]_{E_\ell^{\mathcal{A}\mathcal{T}_A}}\right) \iff \left(a_j = a_k \ \wedge \ i = \ell\right) \tag{4.3}$$

Given this syntactical definition, we let $T_A = \text{range}(h_A) \subseteq \{[a]_{E_i^{\mathcal{A}\mathcal{T}_A}} : a \in A \wedge 0 \leq i \leq n+1\}$.

Lemma 4.12. *The function $h_A : T \to T_A$ as defined in Lemma 4.11 is both 1-1 and onto, and furthermore $\{[a]_{E_{n+1}^{\mathcal{A}\mathcal{T}_A}} : a \in A\} \subseteq T_A$.*

Proof. Since we defined $T_A = \text{range}(h_A)$, it is clear that $h_A : T \to T_A$ is onto. To show then that all of A is covered in T_A, we fix some A-computable enumeration of $A = \{a_0 < a_1 < a_2 < \cdots\}$ and some $\mathcal{T}$-computable enumeration of end nodes $T_{n+1} = \{e_0 < e_1 < e_2 < \cdots\}$ as in Lemma 4.11. Then, given any $a \in A$, $a = a_j$ for some a_j in this enumeration of A. And by definition, $[a_j]_{E_{n+1}^{\mathcal{A}\mathcal{T}_A}} = h_A(e_j)$. Hence for any $a \in A$, $\{[a]_{E_{n+1}^{\mathcal{A}\mathcal{T}_A}} \in \text{range}(h_A)$.

To show that h_A is 1-1 we let x, y be nodes in $\mathcal{T}$ such that $h(x) = h(y)$. Then $h(x) = [a_j]_{E_{\text{level}_{\mathcal{T}}(x)}^{\mathcal{A}\mathcal{T}_A}}$ and $h(y) = [a_k]_{E_{\text{level}_{\mathcal{T}}(y)}^{\mathcal{A}\mathcal{T}_A}}$ for some $a_j, a_k \in A = \{a_0 < a_1 < a_2 < \cdots\}$ where e_j is the least terminating node that is $\succeq_{\mathcal{T}} x$ and e_k is the least terminating node that is $\succeq_{\mathcal{T}} y$. Since $h(x) = h(y)$, this implies that $e_j = e_k$ and $\text{level}_{\mathcal{T}}(x) = \text{level}_{\mathcal{T}}(y)$ as in Equation 4.3. Therefore x and y are both predecessors of $e_j = e_k$. Since $\mathcal{T}$ is a tree, $\{x, y\}$ is a well-ordered set, and hence must have a least element. But $\text{level}_{\mathcal{T}}(x) = \text{level}_{\mathcal{T}}(y)$, which implies that $x \not\prec_{\mathcal{T}} y$ and $y \not\prec_{\mathcal{T}} x$. Therefore the only way that this is possible is if $x = y$. □

Since h_A simply relabels the nodes of $\mathcal{T}$ in a unique, 1-1 way, we can define a tree, $\mathcal{T}_A = (T_A, \prec_{\mathcal{T}_A})$ where $\prec_{\mathcal{T}_A}$ is inherited directly from $\prec_{\mathcal{T}}$. That is,

$$h_A(x) \prec_{\mathcal{T}_A} h_A(y) \overset{\text{defn}}{\iff} x \prec_{\mathcal{T}} y \tag{4.4}$$

Or equivalently,

$$[a_j]_{E_i^{\mathcal{A}\mathcal{T}_A}} \prec_{\mathcal{T}_A} [a_k]_{E_\ell^{\mathcal{A}\mathcal{T}_A}} \overset{\text{defn}}{\iff} h_A^{-1}\big([a_j]_{E_i^{\mathcal{A}\mathcal{T}_A}}\big) \prec_{\mathcal{T}} h_A^{-1}\big([a_k]_{E_\ell^{\mathcal{A}\mathcal{T}_A}}\big) \tag{4.5}$$

We now prove what is intuitively fairly obvious – that $\mathcal{T}_A$ is indeed a tree and furthermore it is isomorphic to $\mathcal{T}$. We also prove that the construction is $(\mathcal{T} \oplus A)$-computable. For the following lemma, assume we are given $\mathcal{T}$ as in Theorem 4.10. Then let $\mathcal{T}_A = (T_A, \prec_{\mathcal{T}_A})$ with $T_A = \text{range}(h_A)$ for h_A defined as in Equation 4.2, and with $\prec_{\mathcal{T}_A}$ defined as in Equation 4.4 and Equation 4.5.

Lemma 4.13. *$\mathcal{T}_A$ is a $(\mathcal{T} \oplus A)$-computable tree, and furthermore $\mathcal{T}_A$ is $(\mathcal{T} \oplus A)$-computably isomorphic to $\mathcal{T}$.*

Proof. We must first show that $\mathcal{T}_A$ as defined is indeed a tree. Since $T_A = \text{range}(h_A)$, and we showed h_A was 1-1, then T_A is a countably infinite set since T is. Also, $\prec_{\mathcal{T}_A}$ as defined is a strict partial

order on T_A:

- Irreflexive: Let $x \in T_A$. This implies that $h_A^{-1}(x) \in \mathcal{T}$.

$$\begin{aligned} \mathcal{T} \text{ is a tree } &\implies h_A^{-1}(x) \not\prec_{\mathcal{T}} h_A^{-1}(x) \\ &\iff x \not\prec_{\mathcal{T}_A} x, \text{ as in Equation 4.4 and Equation 4.5} \end{aligned}$$

- Transitive: Let $x, y, z \in T_A$. Then $h_A^{-1}(x)$, $h_A^{-1}(y)$, $h_A^{-1}(z) \in \mathcal{T}$, and we have:

$$\begin{aligned} x \prec_{\mathcal{T}_A} y \ \wedge \ y \prec_{\mathcal{T}_A} z &\iff h_A^{-1}(x) \prec_{\mathcal{T}} h_A^{-1}(y) \ \wedge \ h_A^{-1}(x) \prec_{\mathcal{T}} h_A^{-1}(y) \\ &\implies h_A^{-1}(x) \prec_{\mathcal{T}} h_A^{-1}(z) \quad (\text{since } \mathcal{T} \text{ a tree, hence } \prec_{\mathcal{T}} \text{ is transitive}) \\ &\iff x \prec_{\mathcal{T}_A} z \quad (\text{by definition}) \end{aligned}$$

- Assymmetric: Let $x, y \in T_A$. Then $h_A^{-1}(x)$, $h_A^{-1}(y) \in \mathcal{T}$, and we have:

$$x \prec_{\mathcal{T}_A} y \iff h_A^{-1}(x) \prec_{\mathcal{T}} h_A^{-1}(y) \implies h_A^{-1}(y) \not\prec_{\mathcal{T}} h_A^{-1}(x) \iff y \not\prec_{\mathcal{T}_A} x$$

To show that predecessors are well-ordered, we let $\{y_1, \ldots, y_m\}$ be predecessors under $\prec_{\mathcal{T}_A}$ of some $x \in T_A$. By the way we've defined $\prec_{\mathcal{T}_A}$ this means that $\{h_A^{-1}(y_1), \ldots, h_A^{-1}(y_m)\}$ are predecessors of $h_A^{-1}(x)$ on our tree $\mathcal{T} = (T, \prec_{\mathcal{T}})$, and hence has a least element, call it $h_A^{-1}(y_{m_0})$. Therefore, for each $\ell \in \{1, \ldots, m\} - \{m_0\}$,

$$h_A^{-1}(y_{m_0}) \prec_{\mathcal{T}} h_A^{-1}(y_\ell) \iff y_{m_0} \prec_{\mathcal{T}_A} y_\ell$$

Hence, y_{m_0} is the least element of $\{y_1, \ldots, y_m\}$ under $\prec_{\mathcal{T}_A}$, and the set of predecessors of x must therefore be well-ordered.

We now must show that $\mathcal{T}_A$ is a $(\mathcal{T} \oplus A)$-computable tree. First we show that T_A is a $(\mathcal{T} \oplus A)$-computable set. Again, we note that T_A is simply a set of nodes with some certain syntax as defined by h_A. So, given some $[a]_{E_i(\mathcal{T}_A)}$, can we $(\mathcal{T} \oplus A)$-computably tell if $[a]_{E_i(\mathcal{T}_A)} \in T_A = \text{range}(h_A)$? Below is such a process.

1. First ask $a \in A$? This is clearly an A-computable question. If no, then stop; we know $[a]_{E_i^{\mathcal{A}_{\mathcal{T}_A}}} \notin T_A$. If yes, then continue to next step.

2. Then ask if $i \leq n + 1$. If no, then stop; we know $[a]_{E_i^{\mathcal{A}_{\mathcal{T}_A}}} \notin T_A$. If yes, then continue to next

step.

3. Ask if $i = n+1$. If yes, then stop; we know $[a]_{E_i^{\mathcal{A}\mathcal{T}_A}} \in T_A$. If no, then this means that $i < n+1$, continue to next step.

4. Now use the A-oracle to enumerate A as $A = \{a_0, a_1, a_2, \ldots\}$ where $a_0 < a_1 < a_2 < \cdots$. Then find k such that $a = a_k$. (This is an A-computable process.)

5. Now using a $\mathcal{T}$-oracle, find the 0th through kth end nodes, e_0 through e_k, in the $\mathcal{T}$-computable enumeration of end nodes $T_{n+1} = \{e_0, e_1, \ldots\}$ with $e_0 < e_1 < \cdots$ from Lemma 4.8. (This is a $\mathcal{T}$-computable process.) Also find the level i predecessor of e_k in $\mathcal{T}$, which is also $\mathcal{T}$-computable by Lemma 4.5 and Lemma 4.3. Call it $p_{e_k,i}$.

6. Now, for each of $e_0, \ldots, e_{k-1}$, check if $p_{e_k,i}$ is also a predecessor of any of $e_0, \ldots, e_{k-1}$. Conducting this check is $\mathcal{T}$-computable, since $\prec_{\mathcal{T}}$ is $\mathcal{T}$-computable. If no, $p_{e_k,i}$ is not a predecessor for any of $e_0, \ldots, e_{k-1}$ (that is, $p_{e_k,i} \not\prec_{\mathcal{T}} e_0 \ \wedge \ p_{e_k,i} \not\prec_{\mathcal{T}} e_1 \ \wedge \ \cdots \ \wedge \ p_{e_k,i} \not\prec_{\mathcal{T}} e_{k-1}$), then stop; we know that $[a]_{E_i^{\mathcal{A}\mathcal{T}_A}} \in T_A$. If yes, $p_{e_k,i}$ is a predecessor for at least one of $e_0, \ldots, e_{k-1}$ (that is, $p_{e_k,i} \prec_{\mathcal{T}} e_0 \ \vee \ p_{e_k,i} \prec_{\mathcal{T}} e_1 \ \vee \ \cdots \ \vee \ p_{e_k,i} \prec_{\mathcal{T}} e_{k-1}$), then stop; we know that $[a]_{E_i^{\mathcal{A}\mathcal{T}_A}} \notin T_A$.

The above process relied only upon a $\mathcal{T}$-oracle and an A-oracle, therefore T_A is a $(\mathcal{T} \oplus A)$-computable set.

Now, we need to check that $\prec_{\mathcal{T}_A}$ is a $(\mathcal{T} \oplus A)$-computable relation. Given two nodes in $\mathcal{T}_A$, that is, given some $[a]_{E_i^{\mathcal{A}\mathcal{T}_A}}, [b]_{E_\ell^{\mathcal{A}\mathcal{T}_A}} \in T_A$, we describe a $(\mathcal{T} \oplus A)$-computable process to determine whether $[a]_{E_i^{\mathcal{A}\mathcal{T}_A}} \prec_{\mathcal{T}_A} [b]_{E_\ell^{\mathcal{A}\mathcal{T}_A}}$. Recalling our definition of $\prec_{\mathcal{T}_A}$ in Equation 4.5 we proceed as follows.

1. First, A-computably fix some enumeration of $A = \{a_0, a_1, \ldots\}$ such that $a_0 < a_1 < \cdots$ as in Lemma 4.11, and determine which elements a and b correspond to. (That is, find j, k such that $a = a_j$ and $b = a_k$.) This is A-computable.

2. Then using a $\mathcal{T}$-oracle, fix some enumeration of end nodes of $\mathcal{T}$, $T_{n+1} = \{e_0, e_1, \ldots\}$ with $e_0 < e_1 < \cdots$ as in Lemma 4.8, and find the jth and kth end node in this enumeration, e_j and e_k.

3. Next find the ith level predecessor of e_j, call it $p_{e_j,i}$, and find the ℓth level predecessor of e_k, call it $p_{e_k,\ell}$. This is a $\mathcal{T}$-computable process since $P_{e_j} = \{\text{predecessors of } e_j\}$ and $P_{e_k} = \{\text{predecessors of } e_k\}$ are $\mathcal{T}$-computable by Corollary 4.6, and $\text{level}_{\mathcal{T}}$ is $\mathcal{T}$-computable by Corollary 4.4.

4. Now, using our $\mathcal{T}$-oracle, check whether $p_{e_j,i} \prec_{\mathcal{T}_A} p_{e_k,\ell}$. If yes, then stop; we know that $[a]_{E_i^{\mathcal{A}\mathcal{T}_A}} \prec_{\mathcal{T}_A} [b]_{E_\ell^{\mathcal{A}\mathcal{T}_A}}$. If no, then stop; we know that $[a]_{E_i^{\mathcal{A}\mathcal{T}_A}} \not\prec_{\mathcal{T}_A} [b]_{E_\ell^{\mathcal{A}\mathcal{T}_A}}$.

The above process relied only upon a $\mathcal{T}$-oracle and an A-oracle, therefore $\prec_{T_A}$ is a $(\mathcal{T}\oplus A)$-computable relation, and hence $\mathcal{T}_A$ is a $(\mathcal{T} \oplus A)$-computable tree.

Since we already showed that h_A is 1-1 and onto, to show h_A is a $(\mathcal{T} \oplus A)$-computable isomorphism, we need only show that h_A preserves order and h_A is computable. We already proved that h_A is $(\mathcal{T} \oplus A)$-computable in Lemma 4.11. We took preserving order under h_A as definition in Equation 4.4. Therefore $\mathcal{T}_A$ is $(\mathcal{T} \oplus A)$-computably isomorphic to $\mathcal{T}$. □

This concludes the proof of Theorem 4.10. Focusing our examination on computable trees, $\mathcal{T}$, and a computable sets, A, the following corollaries are immediate.

Corollary 4.14. *Given a computable tree $\mathcal{T} = (T, \prec_{\mathcal{T}})$ which is full and has finite height $n+1 \geq 2$, and given any infinite, computable set $A \subseteq \mathbb{N}$, we can define a computable tree $\mathcal{T}_A = (T_A, \prec_{\mathcal{T}_A})$ in which each node is of the form $[a]_{E_i^{\mathcal{A}\mathcal{T}_A}}$ for some $a \in A$ and some $i \in \{0, \ldots, n+1\}$, and furthermore $\mathcal{T}_A$ is computably isomorphic to $\mathcal{T}$.*

Corollary 4.15. *Given a computable tree $\mathcal{T} = (T, \prec_{\mathcal{T}})$ which is full and has finite height $n+1 \geq 2$, we can define a computable tree $\mathcal{T}_{\mathbb{N}} = (T_{\mathbb{N}}, \prec_{\mathcal{T}_{\mathbb{N}}})$ in which each node is of the form $[a]_{E_i^{\mathcal{A}\mathcal{T}_{\mathbb{N}}}}$ for some $a \in \mathbb{N}$ and some $i \in \{0, \ldots, n+1\}$, and furthermore $\mathcal{T}_{\mathbb{N}}$ is computably isomorphic to $\mathcal{T}$.*

Before moving on, we note a few nice properties of the nodes of T_A as built. These properties will be quite useful as we complete our construction of a nested equivalence structure.

Corollary 4.16. *Given $\mathcal{T}$ as in Theorem 4.10, let $\mathcal{T}_A = (T_A, \prec_{\mathcal{T}_A})$ with $T_A = range(h_A)$ for h_A defined as in Lemma 4.11, and with $\prec_{\mathcal{T}_A}$ defined as in Equation 4.4 and Equation 4.5. Then the following properties hold:*

1. $level_{\mathcal{T}_A}([a_j]_{E_i^{\mathcal{A}\mathcal{T}_A}}) = i$
2. $\forall j \in \mathbb{N} \left([a_j]_{E_{n+1}^{\mathcal{A}\mathcal{T}_A}} \in T_A \right)$
3. $[a_j]_{E_i^{\mathcal{A}\mathcal{T}_A}} \in T_A \implies [a_j]_{E_i^{\mathcal{A}\mathcal{T}_A}} \preceq_{\mathcal{T}_A} [a_j]_{E_{n+1}^{\mathcal{A}\mathcal{T}_A}}$
4. $[a_j]_{E_i^{\mathcal{A}\mathcal{T}_A}} \prec_{\mathcal{T}_A} [a_k]_{E_\ell^{\mathcal{A}\mathcal{T}_A}} \implies a_j \leq a_k$
5. $[a_0]_{E_0^{\mathcal{A}\mathcal{T}_A}}$ *is the root node*

Proof. This is all straightforward by construction.

1. By construction $h_A^{-1}([a_j]_{E_i^{\mathcal{A}\mathcal{T}_A}}) = p_{e_j,i}$ = the level i predecessor of e_j. Since h_A is an isomorphism, it therefore preserves levels of the tree. Hence, $\text{level}_{\mathcal{T}_A}([a_j]_{E_i^{\mathcal{A}\mathcal{T}_A}}) = i$

2. This is just another way of rewording the final part of Lemma 4.12.

3. Since $T_A = \text{range}(h_A)$, we know that $h_A^{-1}([a_j]_{E_i^{\mathcal{A}\mathcal{T}_A}}) = p_{e_j,i}$, which is defined to be the level i predecessor of e_j. By construction we also know that $h_A(e_j) = [a_j]_{E_{n+1}^{\mathcal{A}\mathcal{T}_A}}$. Therefore,

$$\begin{aligned} p_{e_j,i} \preceq_{\mathcal{T}} e_j &\iff h_A^{-1}([a_j]_{E_i^{\mathcal{A}\mathcal{T}_A}}) \preceq_{\mathcal{T}_A} h_A^{-1}([a_j]_{E_{n+1}^{\mathcal{A}\mathcal{T}_A}}) \\ &\iff [a_j]_{E_i^{\mathcal{A}\mathcal{T}_A}} \preceq_{\mathcal{T}_A} [a_j]_{E_{n+1}^{\mathcal{A}\mathcal{T}_A}} \quad \text{(by Equation 4.5)} \end{aligned}$$

4. We assume that $[a_j]_{E_i^{\mathcal{A}\mathcal{T}_A}} \prec_{\mathcal{T}_A} [a_k]_{E_\ell^{\mathcal{A}\mathcal{T}_A}}$. This gives,

$$\begin{aligned} [a_j]_{E_i^{\mathcal{A}\mathcal{T}_A}} \prec_{\mathcal{T}_A} [a_k]_{E_\ell^{\mathcal{A}\mathcal{T}_A}} &\implies [a_j]_{E_i^{\mathcal{A}\mathcal{T}_A}} \prec_{\mathcal{T}_A} [a_k]_{E_\ell^{\mathcal{A}\mathcal{T}_A}} \preceq_{\mathcal{T}_A} [a_k]_{E_{n+1}^{\mathcal{A}\mathcal{T}_A}} \quad \text{(by 3)} \\ &\implies p_{e_j,i} \text{ is the } i\text{th level predecessor of } e_k \\ &\qquad \text{and } h_A(p_{e_k,i}) = [a_j]_{E_i^{\mathcal{A}\mathcal{T}_A}} \end{aligned}$$

   By construction this happens exactly when $e_j \leq e_k$. Therefore, $j \leq k$, since the end nodes were enumerated in order starting with the smallest: $e_0 < e_1 < \cdots$. Since A was also enumerated in order with $a_0 < a_1 < \cdots$, this means that $a_j \leq a_k$, as desired.

5. We begin by referencing the first step in the construction of h_A in Lemma 4.11. This gives us that $[a_0]_{E_0^{\mathcal{A}\mathcal{T}_A}} \in T_A$. By statement 1, $\text{level}_{\mathcal{T}_A}([a_0]_{E_0^{\mathcal{A}\mathcal{T}_A}}) = 0$. Since a tree can only have one node at level 0, $[a_0]_{E_0^{\mathcal{A}\mathcal{T}_A}}$ must be the root node of $\mathcal{T}_A$.

□

Now that we have taken a full, finite height tree and computably relabelled its nodes so that they have the same syntax as equivalence classes, we now have the framework and foundation on which to take these syntactically defined nodes and turn them into a nested equivalence structure. We will use the structure of the tree and the properties of its partial order to define the equivalence classes, the associated equivalence relations, and their nesting properties.

Theorem 4.17. *Given $\mathcal{T}_A$ as built in Theorem 4.10 (a full tree of finite height $n+1 \geq 2$ with nodes T_A of the form $[a_j]_{E_i^{\mathcal{A}\mathcal{T}_A}}$ for some $A = \{a_0 < a_1 < \cdots\} \subseteq \mathbb{N}$ and $i \in \{0, \ldots, n+1\}$ and satisfying the properties of Corollary 4.16), we can define a 1-1 and onto $(\mathcal{T} \oplus A)$-computable*

function $\tilde{h}$ which takes nodes of $\mathcal{T}_A$ and turns them into equivalence classes of A. These equivalence classes are nested and therefore define the equivalence relations of a nested equivalence structure $\mathcal{A}_{\mathcal{T}_A} = (A, E_1^{\mathcal{A}_{\mathcal{T}_A}}, \ldots, E_n^{\mathcal{A}_{\mathcal{T}_A}})$. Furthermore, this nested equivalence structure is itself $(\mathcal{T} \oplus A)$-computable.

Proof. The proof is contained in the following lemmas: Lemma 4.18, Lemma 4.19, Lemma 4.20, and Lemma 4.21. □

We describe now the function $\tilde{h}$ in detail. This function takes the purely syntactically defined nodes of $\mathcal{T}_A$ which look like equivalence classes, and translates them into actual equivalence classes on A. This is the first step in taking our tree $\mathcal{T}_A$ of height $n+1$ and translating it into a nested equivalence structure of the form $\mathcal{A} = (A, E_1^{\mathcal{A}_{\mathcal{T}_A}}, \ldots, E_n^{\mathcal{A}_{\mathcal{T}_A}})$.

Lemma 4.18. *Given $\mathcal{T}_A$ as built in Theorem 4.10, we can define a 1-1 and onto function $\tilde{h}$ which takes nodes of $\mathcal{T}_A$ and turns them into equivalence classes of A.*

Proof. We first note that as built the elements of T_A already look like they represent equivalence classes of the equivalence relations $E_0^{\mathcal{A}_{\mathcal{T}_A}}, E_1^{\mathcal{A}_{\mathcal{T}_A}}, \ldots, E_n^{\mathcal{A}_{\mathcal{T}_A}}, E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}$. We therefore define our function $\tilde{h}$ so as to take the purely syntactical representation of nodes $[a_j]_{E_i}^{\mathcal{A}_{\mathcal{T}_A}} \in T_A$ and turn them into identical symbols, but now with the usual equivalence class meaning given appropriate definitions of relations $E_0^{\mathcal{A}_{\mathcal{T}_A}}, E_1^{\mathcal{A}_{\mathcal{T}_A}}, \ldots, E_n^{\mathcal{A}_{\mathcal{T}_A}}, E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}$. We then let $\tilde{h}$ = the "identity" function on different universes. That is, $\tilde{h}([a_j]_{E_i^{\mathcal{A}_{\mathcal{T}_A}}}) = [a_j]_{E_i^{\mathcal{A}_{\mathcal{T}_A}}}$, where $\tilde{h} : T_A \to C_{\mathcal{A}_{\mathcal{T}_A}}$. For now, we take $C_{\mathcal{A}_{\mathcal{T}_A}}$ to simply be the set of equivalence classes as defined by the equivalence-class-like elements of T_A. In Lemma 4.20, we will show that this set $C_{\mathcal{A}_{\mathcal{T}_A}}$ actually corresponds to the set of equivalence classes as in Lemma 4.1. Clearly as defined, then, $\tilde{h}$ is 1-1 and onto.

To go from the syntactical set of nodes T_A to a set of equivalence classes, we need only define the elements of these different classes. We do this by defining the following equivalence relations for $i \in \{0, \ldots, n+1\}$:

$$aE_i^{\mathcal{A}_{\mathcal{T}_A}} b \overset{\text{defn}}{\iff} \exists \text{ node } x \text{ at level } i \text{ of } T_A \text{ s.t. } [a]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}} \succeq_{\mathcal{T}_A} x \text{ and } [b]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}} \succeq_{\mathcal{T}_A} x \tag{4.6}$$

Now, we need to show that each $E_i^{\mathcal{A}_{\mathcal{T}_A}}$ is indeed an equivalence relation on A for $i \in \{0, \ldots, n+1\}$. We let $a, b, c \in A$ and $i \in \{0, \ldots, n\}$. We have:

- Reflexive: Note that $[a]_{E_i^{\mathcal{A}_{\mathcal{T}_A}}} \preceq_{\mathcal{T}_A} [a]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$ for $i \leq n+1$. Therefore there exists some x at level i, namely $x = [a]_{E_i^{\mathcal{A}_{\mathcal{T}_A}}}$, such that $x = [a]_{E_i^{\mathcal{A}_{\mathcal{T}_A}}} \preceq_{\mathcal{T}_A} [a]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$ and $x = [a]_{E_i^{\mathcal{A}_{\mathcal{T}_A}}} \preceq_{\mathcal{T}_A} [a]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$. Therefore $aE_i^{\mathcal{A}_{\mathcal{T}_A}} a$ for $i \in \{0, \ldots, n+1\}$.

- Symmetric: $aE_i^{\mathcal{A}_{\mathcal{T}_A}} b \iff \exists$ node x at level i s.t. $[a]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}} \succeq_{\mathcal{T}_A} x$ and $[b]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}} \succeq_{\mathcal{T}_A} x$ $\iff \exists$ node x at level i s.t. $[b]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}} \succ x$ and $[a]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}} \succ x \iff bE_i^{\mathcal{A}_{\mathcal{T}_A}} a$

- Transitive:

$$
\begin{aligned}
aE_i^{\mathcal{A}_{\mathcal{T}_A}} b \wedge bE_i^{\mathcal{A}_{\mathcal{T}_A}} c \implies & \big(\exists x \text{ at level } i \text{ s.t. } [a]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}} \succeq_{\mathcal{T}_A} x \text{ and } [b]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}} \succeq_{\mathcal{T}_A} x\big), \text{ and} \\
& \big(\exists x \text{ at level } i \text{ s.t. } [b]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}} \succeq_{\mathcal{T}_A} x \text{ and } [c]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}} \succeq_{\mathcal{T}_A} x\big) \\
\implies & \big(\exists x, y \text{ at level } i \text{ s.t. } [a]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}} \succeq_{\mathcal{T}_A} x \text{ and } [b]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}} \succeq_{\mathcal{T}_A} x \text{ and} \\
& [b]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}} \succeq_{\mathcal{T}_A} y \text{ and } [c]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}} \succeq_{\mathcal{T}_A} y\big) \\
\implies & x = y \, (*) \\
& \text{hence } \big(\exists x \text{ at level } i \text{ s.t. } [a]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}} \succeq_{\mathcal{T}_A} x \text{ and } [c]_{E_{n+1}(\mathcal{T}_A)} \succeq_{\mathcal{T}_A} x\big) \\
\implies & aE_i^{\mathcal{A}_{\mathcal{T}_A}} c
\end{aligned}
$$

To finish out (*), we first let $i = n+1$. Since $[b]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$ is at level $n+1$ of $\mathcal{T}_A$ and so are x and y, we must have that $[b]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}} = x$ and $[b]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}} = y$. Therefore $x = y$ as desired. Now we let $i < n+1$ and assume to the contrary, that $x \neq y$. This would give that both x and y are predecessors of $[b]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$. Therefore $\{x, y\}$ must contain a least element, since in a tree the set of a predecessors of a given node is well-ordered. But x and y are both at level i of the tree $\mathcal{T}_A$, hence $x \not\preceq_{\mathcal{T}_A} y$ and $y \not\preceq_{\mathcal{T}_A} x$, a contradiction.

Therefore for $i \in \{0, \ldots, n+1\}$, given the definition in Equation 4.6, each $E_i^{\mathcal{A}_{\mathcal{T}_A}}$ is an equivalence relation on A, as desired.

Now finally, we need to show that the equivalence classes that come from the syntactical nodes of T_A are in fact well-defined under the equivalence relations $E_0^{\mathcal{A}_{\mathcal{T}_A}}, \ldots, E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}$. That is, we need to show that two different $E_i^{\mathcal{A}_{\mathcal{T}_A}}$-equivalence class looking nodes from T_A don't overlap when we turn them into actual equivalence classes under the definition in Equation 4.6. We let $[a_j]_{E_i^{\mathcal{A}_{\mathcal{T}_A}}}$ and $[a_k]_{E_i^{\mathcal{A}_{\mathcal{T}_A}}}$ be two different elements of T_A. We assume to the contrary that these two different equivalence classes do overlap, or rather that $a_j E_i^{\mathcal{A}_{\mathcal{T}_A}} a_k$. By part 3 of Corollary 4.16, $[a_j]_{E_i^{\mathcal{A}_{\mathcal{T}_A}}}$ is the ith level predecessor of $[a_j]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$ and $[a_k]_{E_i^{\mathcal{A}_{\mathcal{T}_A}}}$ is the ith level predecessor of $[a_k]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$. By Equation 4.6, $a_j E_i^{\mathcal{A}_{\mathcal{T}_A}} a_k$ means that there exists some node x at level i of $\mathcal{T}_A$ such that $x \preceq_{\mathcal{T}_A} [a_j]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$ and $x \preceq_{\mathcal{T}_A} [a_k]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$. Since $\mathcal{T}_A$ is a tree, a given node can have at most one predecessor at each level. Therefore $x = [a_j]_{E_i^{\mathcal{A}_{\mathcal{T}_A}}}$ and $x = [a_k]_{E_i^{\mathcal{A}_{\mathcal{T}_A}}}$. Therefore $[a_j]_{E_i^{\mathcal{A}_{\mathcal{T}_A}}} = [a_k]_{E_i^{\mathcal{A}_{\mathcal{T}_A}}} \in T_A$, a contradiction

since we assumed these were different nodes in T_A. $\square$

We next show that these equivalence relations nest exactly as we would hope in order to build a nested equivalence structure. Recalling back to Section 4.3 and Example 4.9, we took an n-nested equivalence structure and created two additional equivalence relations which also nicely nested into that structure: equality and the trivial equivalence relation in which everything is equal. Our construction thus far has resulted in the same.

Lemma 4.19. *Let $E_0^{\mathcal{A}_{\mathcal{T}_A}}, E_1^{\mathcal{A}_{\mathcal{T}_A}}, \ldots, E_n^{\mathcal{A}_{\mathcal{T}_A}}, E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}$ be equivalence relations as defined in Equation 4.6 of Lemma 4.18. Let $\mathcal{A}_{\mathcal{T}_A} = (A, E_1^{\mathcal{A}_{\mathcal{T}_A}}, \ldots, E_n^{\mathcal{A}_{\mathcal{T}_A}})$. Then $\mathcal{A}_{\mathcal{T}_A}$ is a nested equivalence structure and $E_0^{\mathcal{A}_{\mathcal{T}_A}}$ and $E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}$ are also nested as expected. Furthermore, $E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}$ corresponds to the relation of equality, and $E_0^{\mathcal{A}_{\mathcal{T}_A}}$ corresponds to the equivalence relation under which everything in A is equal.*

Proof. Since Lemma 4.18 showed that $E_0^{\mathcal{A}_{\mathcal{T}_A}}, E_1^{\mathcal{A}_{\mathcal{T}_A}}, \ldots, E_n^{\mathcal{A}_{\mathcal{T}_A}}, E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}$ are indeed equivalence relations on A, we now need only show that the $E_i^{\mathcal{A}_{\mathcal{T}_A}}$'s are nested as desired so that: $E_{i+1}^{\mathcal{A}_{\mathcal{T}_A}} \subseteq E_i^{\mathcal{A}_{\mathcal{T}_A}}$ for $i \in \{0, \ldots, n\}$. Recall that $E_{i+1}^{\mathcal{A}_{\mathcal{T}_A}} \subseteq E_i^{\mathcal{A}_{\mathcal{T}_A}} \iff$ for any $a \in A$, $[a]_{E_{i+1}^{\mathcal{A}_{\mathcal{T}_A}}} \subseteq [a]_{E_i^{\mathcal{A}_{\mathcal{T}_A}}}$, where this is referring to actual equivalence classes under $E_i^{\mathcal{A}_{\mathcal{T}_A}}$ and $E_{i+1}^{\mathcal{A}_{\mathcal{T}_A}}$, not just a syntactical node in $\mathcal{T}_A$. To do this, we need to show $\forall\, a_j, a_k \in A\big(a_k E_{i+1}^{\mathcal{A}_{\mathcal{T}_A}} a_j \implies a_k E_i^{\mathcal{A}_{\mathcal{T}_A}} a_j\big)$. So let $a_j, a_k \in A$.

$$a_k E_{i+1}^{\mathcal{A}_{\mathcal{T}_A}} a_j \iff \exists \text{ node } x \text{ at level } i \text{ s.t. } x \preceq_{\mathcal{T}_A} [a_k]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}} \text{ and } x \preceq_{\mathcal{T}_A} [a_j]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$$

Since $[a_j]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$ and $[a_k]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$ are both nodes on $\mathcal{T}_A$, they must each have predecessors at levels $0-n$ of $\mathcal{T}_A$. In particular, they have predecessors at level i of $\mathcal{T}_A$. In other words, there exist y and z at level i such that:

$$y \preceq_{\mathcal{T}_A} [a_k]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}} \text{ and } z \preceq_{\mathcal{T}_A} [a_j]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$$

Therefore $\{x, y\}$ are predecessors of $[a_k]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$ and $\{x, z\}$ are predecessors of $[a_j]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$ (where we mean not necessarily a strict predecessor, but a predecessor under "$\preceq_{\mathcal{T}_A}$"). Both of these sets must be well-ordered sets since $\mathcal{T}_A$ is a tree. Since $\text{level}_{\mathcal{T}_A}(x) = i+1$ and $\text{level}_{\mathcal{T}_A}(y) = \text{level}_{\mathcal{T}_A}(z) = i < i+1$, we must have that:

$$y \prec_{\mathcal{T}_A} x \text{ and } z \prec_{\mathcal{T}_A} x$$

Therefore $\{y, z\}$ are predecessors of x and must also be a well-ordered set. But y and z are both at level i of $\mathcal{T}_A$, which implies $y \not\prec_{\mathcal{T}_A} z$ and $z \not\prec_{\mathcal{T}_A} y$. Therefore the only way this can happen is if

$y = z$. Therefore:

$$y \prec_{\mathcal{T}_A} [a_k]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}} \text{ and } y \prec_{\mathcal{T}_A} [a_k]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}} \text{ with } \text{level}_{\mathcal{T}_A}(y) = i$$

Therefore $a_k E_i^{\mathcal{A}_{\mathcal{T}_A}} a_j$, as desired. Since a_j, a_k were chosen arbitrarily, this yields that $E_{i+1}^{\mathcal{A}_{\mathcal{T}_A}} \subseteq E_i^{\mathcal{A}_{\mathcal{T}_A}}$ for $i \in \{0, \ldots, n\}$.

Now examining $E_0^{\mathcal{A}_{\mathcal{T}_A}}$, first we note that by Corollary 4.16, $[a_0]_{E_0^{\mathcal{A}_{\mathcal{T}_A}}}$ is the root node of $\mathcal{T}_A$. Therefore we know that for any $a_j \in A$, $[a_0]_{E_0^{\mathcal{A}_{\mathcal{T}_A}}} \preceq_{\mathcal{T}_A} [a_j]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$. Additionally, by Corollary 4.16, $[a_0]_{E_0^{\mathcal{A}_{\mathcal{T}_A}}} \preceq_{\mathcal{T}_A} [a_0]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$. This implies $a_0 E_i^{\mathcal{A}_{\mathcal{T}_A}} a_j$ for any $a_j \in A$. Therefore $E_0^{\mathcal{A}_{\mathcal{T}_A}}$ corresponds to the equivalence relation under which everything in A is equal.

Under $E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}$, for any $a_j, a_k \in A$, $a_j E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}} a_k \iff \exists$ node x at level $n+1$ of $\mathcal{T}_A$ such that $x \preceq_{\mathcal{T}_A} [a_j]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$ and $x \preceq_{\mathcal{T}_A} [a_k]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$. Since both $[a_j]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$ and $[a_k]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$ are at level $n+1$ of $\mathcal{T}_A$ by Corollary 4.16, neither one of these nodes can have a strict predecessor that is also at level $n+1$. Therefore $\exists$ node x at level $n+1$ of $\mathcal{T}_A$ such that $x = [a_j]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$ and $x = [a_k]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$. As nodes in $\mathcal{T}_A$ (and not as equivalence classes of $\mathcal{A}_{\mathcal{T}_A}$), x represents only one single node, and has a unique representation, hence $a_j = a_k$. Therefore for any $a_j, a_k \in A$, $a_j E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}} a_k \implies a_j = a_k$, and $E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}$ corresponds to the equivalence relation of equality.

□

In Lemma 4.18 we let $C_{\mathcal{A}_{\mathcal{T}_A}}$ simply represent some set of equivalence classes, specifically equivalence classes with the same syntax as the nodes of $\mathcal{T}_A$. Additionally, we proved that each equivalence class in this set was indeed unique, and therefore our construction was well-defined. In Lemma 4.1, though, we also built a unique representation of equivalence classes of a nested equivalence structure and called it $C_{\mathcal{A}}$. Our notation in Lemma 4.18 was no coincidence, and now that we have built our nested equivalence structure $\mathcal{A}_{\mathcal{T}_A}$, we can take this concept one step further and prove that T_A and $C_{\mathcal{A}_{\mathcal{T}_A}}$ are indeed the "same" sets. (They are identical in terms of notation, of course, but we still think of elements of T_A simply as strings of symbols, whereas elements of $C_{\mathcal{A}_{\mathcal{T}_A}}$ give a one to one representation of equivalence classes.)

For the next lemma, we let $C_{\mathcal{A}_{\mathcal{T}_A}}$ be the enumeration of $\mathcal{A}_{\mathcal{T}_A}$-equivalence classes without repetition, as defined in Lemma 4.1, with the addition of equivalence classes $E_0^{\mathcal{A}_{\mathcal{T}_A}}$ and $E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}$. We let T_A be as defined in Theorem 4.10.

Lemma 4.20. *$C_{\mathcal{A}_{\mathcal{T}_A}}$ and T_A represent syntactically the same set of elements, and are furthermore $(\mathcal{T} \oplus A)$-computable sets.*

Proof. We will show here that $\tilde{h} : T_A \to C_{\mathcal{A}_{\mathcal{T}_A}}$ is indeed "onto" in the sense we intended in Lemma 4.18. We will prove here that if we created T_A via our algorithm in Lemma 4.11 and we created $C_{\mathcal{A}_{\mathcal{T}_A}}$ via our algorithm in Lemma 4.1, then T_A and $C_{\mathcal{A}_{\mathcal{T}_A}}$ yield the exact same set of elements. Note that both algorithms depended upon fixing some (A-computable) enumeration of A, where $A = \{a_0 < a_1 < a_2 < \cdots\}$. Note also that the algorithm in Lemma 4.1 was given for equivalence relations $E_1, \ldots, E_n$ of a nested equivalence structure $\mathcal{A}$. Given results of Lemma 4.19, we can easily extend the algorithm to equivalence relations $E_0^{\mathcal{A}_{\mathcal{T}_A}}, \ldots, E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}$ of our nested equivalence structure $\mathcal{A}_{\mathcal{T}_A}$.

First, we examine the algorithm to create T_A. A detailed examination of the steps of that algorithm gives the following.

$$\begin{aligned}
[a_j]_{E_i^{\mathcal{A}_{\mathcal{T}_A}}} \in T_A \iff & \text{ } p_{e_j,i} \text{ has no end-node successors which are smaller than } e_j \\
& \text{(where } p_{e_j,i} = i\text{th level predecessor of } e_j, \\
& \text{ and } e_j = \text{the } j\text{th smallest end-node of } \mathcal{T}) \\
\iff & \big(x \preceq_{\mathcal{T}} e_j \ \wedge \ \text{level}_{\mathcal{T}}(x) = i \ \wedge \ x \preceq_{\mathcal{T}} e_k\big) \implies e_j \leq e_k
\end{aligned}$$

Now, we examine the algorithm to create $C_{\mathcal{A}_{\mathcal{T}_A}}$. The details of that algorithm, extended to include equivalence relations $E_0^{\mathcal{A}_{\mathcal{T}_A}}$ and $E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}$ nested appropriately, give the following.

$$\begin{aligned}
[a_j]_{E_i^{\mathcal{A}_{\mathcal{T}_A}}} \in C_{\mathcal{A}_{\mathcal{T}_A}} & \iff a_j \text{ is the smallest element of the equivalence class } [a_j]_{E_i^{\mathcal{A}_{\mathcal{T}_A}}} \\
& \iff \big(a_k E_i^{\mathcal{A}_{\mathcal{T}_A}} a_j \implies a_j \leq a_k\big) \\
& \iff \Big(\big(\exists \text{ node } x \text{ at level } i \text{ of } \mathcal{T}_A \text{ s.t. } x \preceq_{\mathcal{T}_A} [a_k]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}} \ \wedge \ x \preceq_{\mathcal{T}_A} [a_j]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}\big) \\
& \qquad \implies a_j \leq a_k\Big) \qquad \text{(applying defn of } E_i^{\mathcal{A}_{\mathcal{T}_A}} \text{ from Equation 4.6)} \\
& \iff \big(\text{level}_{\mathcal{T}_A}(x) = i \ \wedge \ x \preceq_{\mathcal{T}_A} [a_k]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}} \ \wedge \ x \preceq_{\mathcal{T}_A} [a_j]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}\big) \implies a_j \leq a_k \\
& \iff \big(\text{level}_{\mathcal{T}}(h_A^{-1}(x)) = i \ \wedge \ h_A^{-1}(x) \preceq_{\mathcal{T}} e_k \ \wedge \ h_A^{-1}(x) \preceq_{\mathcal{T}_A} e_j\big) \implies e_j \leq e_k \\
& \qquad \text{(applying defn of } \text{“} \prec_{\mathcal{T}_A} \text{”} \text{ from Equation 4.5)} \\
& \iff \big(\text{level}_{\mathcal{T}}(y) = i \ \wedge \ y \preceq_{\mathcal{T}} e_k \ \wedge \ y \preceq_{\mathcal{T}} e_j\big) \implies e_j \leq e_k \\
& \qquad \text{(because } h_A \text{ is onto)}
\end{aligned}$$

Therefore we have that $[a_j]_{E_i^{\mathcal{A}_{\mathcal{T}_A}}} \in T_A \iff [a_j]_{E_i^{\mathcal{A}_{\mathcal{T}_A}}} \in C_{\mathcal{A}_{\mathcal{T}_A}}$, as desired. Since T_A is a $(\mathcal{T} \oplus \mathcal{A})$-computable set as shown in Lemma 4.13, this means that so too is $C_{\mathcal{A}_{\mathcal{T}_A}}$. $\square$

The final step in our construction of a nested equivalence structure from a tree is to show that this construction is $(\mathcal{T} \oplus A)$-computable.

Lemma 4.21. *The structure $\mathcal{A}_{\mathcal{T}_A} = (A, E_1^{\mathcal{A}_{\mathcal{T}_A}}, \ldots, E_n^{\mathcal{A}_{\mathcal{T}_A}})$ is $(\mathcal{T} \oplus A)$-computable. Hence $\tilde{h}$ is $(\mathcal{T} \oplus A)$-computable, and $C_{\mathcal{A}_{\mathcal{T}_A}}$ is $(\mathcal{T} \oplus A)$-computable. Furthermore the equivalence relations $E_0^{\mathcal{A}_{\mathcal{T}_A}}$ and $E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}$ are also $(\mathcal{T} \oplus A)$-computable.*

Proof. We start by showing that $\mathcal{A}_{\mathcal{T}_A} = (A, E_1^{\mathcal{A}_{\mathcal{T}_A}}, \ldots, E_n^{\mathcal{A}_{\mathcal{T}_A}})$ is $(\mathcal{T} \oplus A)$-computable. Clearly, the universe of $\mathcal{A}_{\mathcal{T}_A}$ is $(\mathcal{T} \oplus A)$-computable. Now to show that for each i, $E_i^{\mathcal{A}_{\mathcal{T}_A}}$ is $(\mathcal{T} \oplus A)$-computable, we need to describe a $(\mathcal{T} \oplus A)$-computable process which takes as input some $a, b \in A$ and some $i \in \{0, \ldots, n+1\}$ and tells whether $aE_i^{\mathcal{A}_{\mathcal{T}_A}} b$. Note again that our nested equivalence structure involves only equivalence relations $E_1^{\mathcal{A}_{\mathcal{T}_A}}, \ldots, E_n^{\mathcal{A}_{\mathcal{T}_A}}$, but the proof extends easily for $E_0^{\mathcal{A}_{\mathcal{T}_A}}$ and $E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}$, so we include them here. Below is such a $(\mathcal{T} \oplus A)$-computable process.

1. First, A-computably fix some enumeration of $A = \{a_0, a_1, \ldots\}$ such that $a_0 < a_1 < \cdots$ as in Lemma 4.11, and determine which elements a and b correspond to. (That is, find j, k such that $a = a_j$ and $b = a_k$.) This is A-computable.

2. We now know that $[a_j]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}, [a_k]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}} \in \mathcal{T}_A$ by Corollary 4.16. So next find the ith level predecessors of $[a_j]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$ and $[a_k]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$, call them x and y respectively. This is a $(\mathcal{T} \oplus A)$-computable process since $P_{[a_j]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}}$ and $P_{[a_k]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}}$ (the sets of predecessors of $[a_j]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$ and $[a_k]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$, respectively) are $(\mathcal{T} \oplus A)$-computable by Corollary 4.6 since $\mathcal{T}_A$ is $(\mathcal{T} \oplus A)$-computable by Lemma 4.13. Additionally, $\text{level}_{\mathcal{T}}$ is $(\mathcal{T} \oplus A)$-computable by Corollary 4.4.

3. Now ask $x = y$?

   - If yes, then stop. We know that there is a node at level i of $\mathcal{T}_A$, namely node $x = y$, such that $x \preceq_{\mathcal{T}_A} [a_j]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$ and $x \preceq_{\mathcal{T}_A} [a_k]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$. Therefore $a_j E_i^{\mathcal{A}_{\mathcal{T}_A}} a_k$ by our definition of $E_i^{\mathcal{A}_{\mathcal{T}_A}}$ in Equation 4.6, and hence we know that $aE_i^{\mathcal{A}_{\mathcal{T}_A}} b$.
   - If no, then stop. We know that there is no node z at level i of $\mathcal{T}_A$ such that $z \preceq_{\mathcal{T}_A} [a_j]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$ and $z \preceq_{\mathcal{T}_A} [a_k]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_A}}}$. (If such a node existed, it would have to equal both x and y, but $x \neq y$.) Therefore $\neg(a_j E_i^{\mathcal{A}_{\mathcal{T}_A}} a_k)$ by our definition of $E_i^{\mathcal{A}_{\mathcal{T}_A}}$ in Equation 4.6, and hence we know that $\neg(aE_i^{\mathcal{A}_{\mathcal{T}_A}} b)$.

Therefore $E_i^{\mathcal{A}_{\mathcal{T}_A}}$ is $(\mathcal{T} \oplus A)$-computable for all $i \in \{0, \ldots, n+1\}$. This gives us that $\mathcal{A}_{\mathcal{T}_A}$ is $(\mathcal{T} \oplus A)$-computable.

Additionally, $C_{\mathcal{A}_{\mathcal{T}_A}}$ is $(\mathcal{T} \oplus A)$-computable since $\mathcal{A}_{\mathcal{T}_A}$ is. Since as previously described, $\tilde{h} : \mathcal{T}_A \to C_{\mathcal{A}_{\mathcal{T}_A}}$ is nothing more than the "identity" function, taking nodes in $\mathcal{T}_A$ to their identical equivalence class in $C_{\mathcal{A}_{\mathcal{T}_A}}$, we have then that $\tilde{h}$ is also a $(\mathcal{T} \oplus A)$-computable function. □

This now completes the proof of Theorem 4.17. Again, we focus our examination on computable trees, $\mathcal{T}$, and computable sets A. The following corollaries are immediate.

Corollary 4.22. *Given a computable tree $\mathcal{T}_A$ as built from a computable tree $\mathcal{T}$ and a computable set A as in Theorem 4.10 and Corollary 4.14, we can define a 1-1 and onto computable function $\tilde{h}$ which takes nodes of $\mathcal{T}_A$ and turns them into equivalence classes. These equivalence classes are nested and therefore define the equivalence relations of a nested equivalence structure $\mathcal{A}_{\mathcal{T}_A} = (A, E_1^{\mathcal{A}_{\mathcal{T}_A}}, \ldots, E_n^{\mathcal{A}_{\mathcal{T}_A}})$. Furthermore, this nested equivalence structure is itself computable.*

Corollary 4.23. *Given a computable tree $\mathcal{T}_{\mathbb{N}}$ as built from a computable tree $\mathcal{T}$ as in Theorem 4.10 and Corollary 4.15, we can define a 1-1 and onto computable function $\tilde{h}$ which takes nodes of $\mathcal{T}_{\mathbb{N}}$ and turns them into equivalence classes. These equivalence classes are nested and therefore define the equivalence relations of a nested equivalence structure $\mathcal{A}_{\mathcal{T}_{\mathbb{N}}} = (\mathbb{N}, E_1^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}, \ldots, E_n^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}})$. Furthermore, this nested equivalence structure is itself computable.*

Finally, we combine the previous two theorems to yield our intended construction of a nested equivalence structure. We can also focus our examination on computable trees and computable sets yielding the corollaries below.

Theorem 4.24. *Given a tree $\mathcal{T} = (T, \prec_{\mathcal{T}})$ which is full and has finite height $n + 1 \geq 2$, and given any infinite set $A \subseteq \mathbb{N}$, we can $(\mathcal{T} \oplus A)$-computably define a nested equivalence structure $\mathcal{A}_{\mathcal{T}_A} = (A, E_1^{\mathcal{A}_{\mathcal{T}_A}}, \ldots, E_n^{\mathcal{A}_{\mathcal{T}_A}})$. Furthermore, this nested equivalence structure is itself $(\mathcal{T} \oplus A)$-computable.*

Corollary 4.25. *Given a computable tree $\mathcal{T} = (T, \prec_{\mathcal{T}})$ which is full and has finite height $n+1 \geq 2$, and given any infinite computable set $A \subseteq \mathbb{N}$, we can computably define a nested equivalence structure $\mathcal{A}_{\mathcal{T}_A} = (A, E_1^{\mathcal{A}_{\mathcal{T}_A}}, \ldots, E_n^{\mathcal{A}_{\mathcal{T}_A}})$. Furthermore, this nested equivalence structure is itself computable.*

Corollary 4.26. *Given a computable tree $\mathcal{T} = (T, \prec_{\mathcal{T}})$ which is full and has finite height $n+1 \geq 2$, we can computably define a nested equivalence structure $\mathcal{A}_{\mathcal{T}_{\mathbb{N}}} = (\mathbb{N}, E_1^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}, \ldots, E_n^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}})$. Furthermore, this nested equivalence structure is itself computable.*

4.5 Nested Equivalence Structures to Trees

In this section we formalize the notions of going from a nested equivalence structure to a tree. Just as in the prior section, the methods we describe here would work equally as well for trees and nested equivalence structures with finite domains. However, we wish to focus our examination on trees and nested equivalence structures with countably infinite domains.

So, given some finitely nested equivalence structure $\mathcal{A} = (A, E_1, \ldots, E_n)$, we now wish to build a tree out of it. We will begin by building a tree out of the equivalence classes as we did with our example in Section 4.3, only this time we will formally and algorithmically define the partial order and the nodes. We will call the function which takes equivalence classes of the nested equivalence structure and turns them into nodes of the tree $\hat{\tilde{h}}$. Next, we will take this tree and create one which is computably isomorphic to it, simply by relabelling the nodes so that they come from $\mathbb{N}$. We will call this isomorphism $\hat{h}$. Below is a picture explaining just what we intend to do. We have built the picture going from "right" to "left" so it is clear how it aligns with Figure 4.3 of Section 4.4.

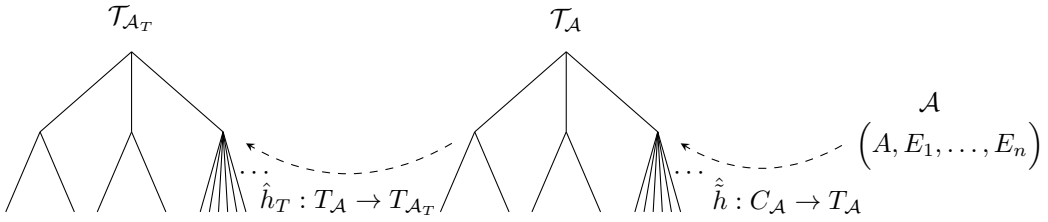


Figure 4.4: Diagram representing the process to take a nested equivalence structure, $\mathcal{A}$, with countably infinite universe and a countably infinite set, T, and build a tree from it.

Note that we have named these two functions $\hat{h}$ and $\hat{\tilde{h}}$ intentionally, because they are certainly closely related to the functions h and $\tilde{h}$ from Section 4.4 where we took a tree and turned it into a nested equivalence structure. Though the functions $\hat{h}$ and $\hat{\tilde{h}}$ behave like inverses of h and $\tilde{h}$, they are not exactly true inverses, as the functions depend upon the base underlying universes of the structures, T and A, and their nesting and ordering properties. In fact, even if we were to ensure that the universes always aligned and applied these functions in appropriate succession, our methods here would not actually give us the exact same structure back that we started with, though it would return one $\deg(\mathcal{A})$-computably isomorphic to it. We will prove this later in Theorem 5.10 and Corollary 5.16.

In all of the steps to go from nested equivalence structures to trees, we will prove that both $\hat{\tilde{h}}$ and $\hat{h}$ are 1-1, onto, computable, and preserve or create the desired structure. We first begin with

$\hat{\tilde{h}}$ in the following theorem.

Theorem 4.27. *Given an n-nested equivalence structure $\mathcal{A} = (A, E_1, \ldots, E_n)$ with infinite domain $A \subseteq \mathbb{N}$, we can define an $\mathcal{A}$-computable, 1-1, and onto function $\hat{\tilde{h}}$ which takes unique equivalence classes of $\mathcal{A}$ and $\mathcal{A}$-computably builds a full tree, $\mathcal{T}_\mathcal{A}$, of height $n+1$ in which each node represents an equivalence class of $\mathcal{A}$.*

Proof. The proof is contained in Lemma 4.28 and Lemma 4.30. □

.

Lemma 4.28. *Let $\mathcal{A} = (A, E_1, \ldots, E_n)$ be a nested equivalence structure with infinite domain $A \subseteq \mathbb{N}$. We can define a function $\hat{\tilde{h}}$ which is $\mathcal{A}$-computable, 1-1, and onto, and takes equivalence classes of $\mathcal{A}$ and turns them into nodes of a tree $\mathcal{T}_\mathcal{A} = (T_\mathcal{A}, \prec_{\mathcal{T}_\mathcal{A}})$.*

Proof. We begin by extending our nested equivalence structure $\mathcal{A}$ to include two additional equivalence relations: E_0 and E_{n+1}. Let E_0 be the relation under which everything is equivalent, and let E_{n+1} be equality. It is easy to see that both are computable equivalence relations and they nest inside of $E_1, \ldots, E_n$ as follows:

$$E_{n+1} \subseteq E_n \subseteq \cdots \subseteq E_1 \subseteq E_0$$

Therefore, we can $\mathcal{A}$-computably fix some enumeration of $A = \{a_0 < a_1 < a_2 < \cdots\}$ and we can extend Lemma 4.1 to create a list of all equivalence classes of A under $E_0, \ldots, E_{n+1}$ without repetition. We call this set $C_\mathcal{A}$. Now, we let $T_\mathcal{A} = C_\mathcal{A}$ in the following sense:

$$[a_j]_{E_i} \in C_\mathcal{A} \iff [a_j]_{E_i} \in T_\mathcal{A}$$

$$[a_j]_{E_i}, [a_k]_{E_\ell} \in T_\mathcal{A} \implies \Big([a_j]_{E_i} = [a_k]_{E_\ell} \iff (i = \ell \wedge j = k)\Big)$$

$$[a_j]_{E_i}, [a_k]_{E_\ell} \in C_\mathcal{A} \implies \Big([a_j]_{E_i} = [a_k]_{E_\ell} \iff (i = \ell \wedge j = k)\Big)$$

$$[a_j]_{E_i}, [a_k]_{E_\ell} \in \{\text{all equiv classes of } \mathcal{A}\} \implies \Big([a_j]_{E_i} = [a_k]_{E_\ell} \iff (i = \ell \wedge a_j E_i a_k)\Big)$$

As with $\tilde{h}$ in Lemma 4.18, we define $\hat{\tilde{h}}$ to be the "identity" function, $\hat{\tilde{h}} : C_\mathcal{A} \to T_\mathcal{A}$. Clearly, then this function is 1-1, onto, and $\mathcal{A}$-computable.

We now take this set $T_\mathcal{A}$ which was derived from the set $C_\mathcal{A}$ of uniquely enumerated equivalence classes of $\mathcal{A}$ without repetition, and turn them into nodes on a tree by defining the tree and the

partial order. Let $\mathcal{T}_{\mathcal{A}} = (T_{\mathcal{A}}, \prec_{\mathcal{T}_{\mathcal{A}}})$ with $\prec_{\mathcal{T}_{\mathcal{A}}}$ and $\preceq_{\mathcal{T}_{\mathcal{A}}}$defined as follows.

$$[a_j]_{E_i} \prec_{\mathcal{T}_{\mathcal{A}}} [a_k]_{E_\ell} \overset{\text{defn}}{\iff} [a_j]_{E_i} \supseteq [a_k]_{E_\ell} \text{ and } i < \ell \tag{4.7}$$

$$[a_j]_{E_i} \preceq_{\mathcal{T}_{\mathcal{A}}} [a_k]_{E_\ell} \overset{\text{defn}}{\iff} [a_j]_{E_i} \supseteq [a_k]_{E_\ell} \text{ and } i \leq \ell \tag{4.8}$$

Note that $[a_j]_{E_i}$, $[a_k]_{E_\ell}$ on the right hand side of the above equations refers to the usual definition of equivalence classes, in which $[a_j]_{E_i} = [a_k]_{E_\ell} \iff (i = \ell \wedge a_j E_i a_k)$. On the other hand, $[a_j]_{E_i}$, $[a_k]_{E_\ell}$ on the left hand side of the above equations refers to (soon to be) nodes on the tree $\mathcal{T}_{\mathcal{A}}$. These nodes have a purely syntactical definition, in which nodes $[a_j]_{E_i} = [a_k]_{E_\ell} \iff (i = \ell \wedge j = k)$.

Now, to prove that $\mathcal{T}_{\mathcal{A}}$ is indeed a tree, we must show that (1) $\prec_{\mathcal{T}_{\mathcal{A}}}$ is a strict partial order on the nodes of $\mathcal{T}_{\mathcal{A}}$, and (2) $\forall x \in T_{\mathcal{A}}$ the set of predecessors of x in $T_{\mathcal{A}}$ is well-ordered by $\prec_{\mathcal{T}_{\mathcal{A}}}$. We begin with (1) and let $x, y, z \in T_{\mathcal{A}}$. By the way we've defined nodes in $\mathcal{T}_{\mathcal{A}}$, this means that $x = [a]_{E_i}$, $y = [b]_{E_j}$, and $z = [c]_{E_k}$ for some $a, b, c \in A$ and some $i, j, k \in \{0, \ldots, n+1\}$.

- Irreflexive: We first note that as equivalence classes in $\mathcal{A}$, $[a]_{E_i} \supseteq [a]_{E_i}$, but in this case we have that $i = i$. This implies $x \not\prec_{\mathcal{T}_{\mathcal{A}}} x$ since in our definition of "$\prec_{\mathcal{T}_{\mathcal{A}}}$" in Equation 4.7, we would need $i < i$.

- Transitive:

$$\begin{aligned} x \prec_{\mathcal{T}_{\mathcal{A}}} y \text{ and } y \prec_{\mathcal{T}_{\mathcal{A}}} z &\iff \big([a]_{E_i} \supseteq [b]_{E_j} \text{ and } i < j\big) \text{ and } \big([b]_{E_j} \supseteq [c]_{E_k} \text{ and } j < k\big) \\ &\iff \big([a]_{E_i} \supseteq [b]_{E_j} \supseteq [c]_{E_k} \text{ and } i < j < k\big) \\ &\implies \big([a]_{E_i} \supseteq [c]_{E_k} \text{ and } i < k\big) \\ &\implies x \prec_{\mathcal{T}_{\mathcal{A}}} z \end{aligned}$$

- Assymmetric:

$$x \prec_{\mathcal{T}_{\mathcal{A}}} y \implies [a]_{E_i} \supseteq [b]_{E_j} \text{ and } i < j \implies j \not< i \implies y \not\prec_{\mathcal{T}_{\mathcal{A}}} x$$

We now show (2), that the sets of predecessors are well-ordered. Let x be some node in $\mathcal{T}_{\mathcal{A}}$, and let P_x represent the set of all predecessors of x in $\mathcal{T}_{\mathcal{A}}$ under $\prec_{\mathcal{T}_{\mathcal{A}}}$. Therefore we have that $x = [a_j]_{E_i}$ for some $a_j \in A$ and $i \in \{0, \ldots, n+1\}$. Since $\mathcal{A}$ is a nested equivalence structure, each E_i-equivalence

class is nested exactly as follows with $[a_j]_{E_i}$ contained in no other E_j-equivalence classes but these:

$$[a_j]_{E_i} \subseteq [a_j]_{E_{i-1}} \subseteq [a_j]_{E_{i-2}} \subseteq \cdots \subseteq [a_j]_{E_1} \subseteq [a_j]_{E_0}$$

Since each of the above equivalence classes are equivalence classes of $\mathcal{A}$, they are each represented exactly once in $C_{\mathcal{A}}$. Furthermore, recalling Equation 4.1, if we let a_{j_k} be the smallest element of $[a_j]_{E_k}$ for each $k \in \{0, 1, \ldots, i-1\}$, we get:

$$[a_j]_{E_i} \subseteq [a_{j_{i-1}}]_{E_{i-1}} \subseteq [a_{j_{i-2}}]_{E_{i-2}} \subseteq \cdots \subseteq [a_{j_1}]_{E_1} \subseteq [a_{j_0}]_{E_0},$$

where each of the above listed equivalence classes is the representative in $C_{\mathcal{A}}$ for the corresponding equivalence class of a_j. Therefore the nodes $[a_{j_0}]_{E_0}$, $[a_{j_1}]_{E_1}$, ... , $[a_{j_{i-1}}]_{E_{i-1}}$ are all also in $\mathcal{T}_{\mathcal{A}}$. Additionally, under the definition of $\prec_{\mathcal{T}_{\mathcal{A}}}$ in Equation 4.7, this $\implies P_x = \{[a_{j_{i-1}}]_{E_{i-1}}, [a_{j_{i-2}}]_{E_{i-2}}, \ldots, [a_{j_1}]_{E_1}, [a_{j_0}]_{E_0}\}$, with the members of P_x ordered as follows:

$$[a_{j_0}]_{E_0} \prec_{\mathcal{T}_{\mathcal{A}}} [a_{j_1}]_{E_1} \prec_{\mathcal{T}_{\mathcal{A}}} \cdots \prec_{\mathcal{T}_{\mathcal{A}}} [a_{j_{i-2}}]_{E_{i-2}} \prec_{\mathcal{T}_{\mathcal{A}}} [a_{j_{i-1}}]_{E_{i-1}}$$

Now, clearly any subset of P_x has a least element – just take $[a_{j_k}]_{E_k}$ for the smallest (under the usual ordering of $\mathbb{N}$) value of k contained in the subset. Therefore P_x is well-ordered, as desired.

Therefore we have built a universe of nodes for $\mathcal{T}_{\mathcal{A}}$ and a strict partial order $\prec_{\mathcal{T}_{\mathcal{A}}}$ on that universe which well-orders the set of predecessors of any given node. Therefore $\mathcal{T}_{\mathcal{A}}$ is indeed a tree. □

Note that in the prior lemma we are abusing notation slightly in reference to $C_{\mathcal{A}}$. As built in Lemma 4.1, $C_{\mathcal{A}}$ represented the set of all equivalence classes of $\mathcal{A}$ enumerated uniquely and without repetition, for $\mathcal{A} = (A, E_1, \ldots, E_n)$. We are still now examining $\mathcal{A} = (A, E_1, \ldots, E_n)$, but we have added two additional computable equivalence relations nested as expected, E_0 and E_{n+1}. In the proofs of these lemmas and theorems, we now use $C_{\mathcal{A}}$ to refer to the list of all equivalence classes of A under $E_0, \ldots, E_{n+1}$ enumerated uniquely and without repetition. The difference in notation — whether $C_{\mathcal{A}}$ refers to equivalence classes under $E_1, \ldots, E_n$ or under $E_0, \ldots, E_{n+1}$ — should be clear from the context.

Corollary 4.29. *Given a nested equivalence structure $\mathcal{A}$ as in Theorem 4.27, let $\mathcal{T}_{\mathcal{A}} = (T_{\mathcal{A}}, \prec_{\mathcal{T}_{\mathcal{A}}})$ where $T_{\mathcal{A}} = C_{\mathcal{A}}$ and $\prec_{\mathcal{T}_{\mathcal{A}}}$ is defined as in Equation 4.7. Then the following properties hold:*

1. $ht(\mathcal{T}_{\mathcal{A}}) = n + 1$

2. *$level_{\mathcal{T}_\mathcal{A}}([a_j]_{E_i}) = i$ for each $[a_j]_{E_i} \in T_\mathcal{A}$*

3. *$\mathcal{T}_\mathcal{A}$ has infinitely many nodes at level $n+1$*

4. *$\mathcal{T}_\mathcal{A}$ is a full tree*

Proof. We fix some enumeration of $A = \{a_0 < a_1 < \cdots\}$. First we prove that $\mathrm{ht}(\mathcal{T}_\mathcal{A}) = n+1$. We know that $[a_0]_{E_0}$, $[a_0]_{E_1}$, ..., $[a_0]_{E_{n+1}}$ are all in $C_\mathcal{A}$ and hence in $T_\mathcal{A}$ since we enumerated them all into $C_\mathcal{A}$ as part of the first step of Lemma 4.1. Because $\mathcal{A}$ is nested we know that:

$$[a_0]_{E_0} \supseteq [a_0]_{E_1} \supseteq \cdots \supseteq [a_0]_{E_n} \supseteq [a_0]_{E_{n+1}}$$

We also note that $0 < 1 < \cdots < n < n+1$. Now, recalling the definition of $\prec_{\mathcal{T}_\mathcal{A}}$ from Equation 4.7, we then have that:

$$[a_0]_{E_0} \prec_{\mathcal{T}_\mathcal{A}} [a_0]_{E_1} \prec_{\mathcal{T}_\mathcal{A}} \cdots \prec_{\mathcal{T}_\mathcal{A}} [a_0]_{E_n} \prec_{\mathcal{T}_\mathcal{A}} [a_0]_{E_{n+1}}$$

Therefore we have created a path of length $n+2$ on our tree $\mathcal{T}_\mathcal{A}$. Therefore $\mathrm{ht}(\mathcal{T}_\mathcal{A}) \geq n+1$. We wish to show that additionally, $\mathrm{ht}(\mathcal{T}_\mathcal{A}) \leq n+1$. Assume to the contrary that $\mathrm{ht}(\mathcal{T}_\mathcal{A}) > n+1$, and hence there exists a path of length $> n+2$. Then we have $(n+3)$-many nodes, call them $x_0, \ldots, x_{n+2} \in T_\mathcal{A}$ such that $x_0 \prec_{\mathcal{T}_\mathcal{A}} \cdots \prec_{\mathcal{T}_\mathcal{A}} x_{n+2}$. Since each $x_j \in T_\mathcal{A}$, we know that each is of the form $x_j = [a_{k_j}]_{E_{\ell_j}}$ where $j \in \{0, \ldots, n+2\}$, $\ell_j \in \{0, \ldots, n+1\}$ and $k_j \in \mathbb{N}$, and as in Equation 4.7 the ℓ_j's must form a chain under "$<$". Without loss of generality, assume $\ell_0 < \ell_1 < \cdots < \ell_{n+1} < \ell_{n+2}$. But each $\ell_j \in \{0, \ldots, n+1\}$, and therefore we have a list of $(n+3)$-many numbers strictly ordered in which any single number can be one of only $(n+2)$-many choices. The pigeon-hole principle states that this is impossible, and we must have at least two of $\ell_0, \ldots, \ell_{n+2}$ being equal, a contradiction. Therefore there is no path in $\mathcal{T}_\mathcal{A}$ of length $> n+2$, and hence $\mathrm{ht}(\mathcal{T}_\mathcal{A}) \leq n+1$. Taking this together we have $\mathrm{ht}(\mathcal{T}_\mathcal{A}) \geq n+1$ and $\mathrm{ht}(\mathcal{T}_\mathcal{A}) \leq n+1$, yielding $\mathrm{ht}(\mathcal{T}_\mathcal{A}) = n+1$.

Now, consider $[a_j]_{E_{n+1}}$ for $j \in \mathbb{N}$. By defining E_{n+1} to be the equivalence relation of "$=$", and by our method of creating $C_\mathcal{A}$ in Lemma 4.1, we know that for each $j \in \mathbb{N}$, $[a_j]_{E_{n+1}} \in C_\mathcal{A} = T_\mathcal{A}$. Because $\mathcal{A}$ is a nested equivalence structure, as equivalence classes we have that:

$$[a_j]_{E_0} \supseteq [a_j]_{E_1} \supseteq \cdots \supseteq [a_j]_{E_n} \supseteq [a_j]_{E_{n+1}}$$

Note however that this is a list of equivalence classes in $\mathcal{A}$, and except for $[a_j]_{E_{n+1}}$ we don't know if each of these is in $C_\mathcal{A}$. What we do know is that each of these $[a_j]_{E_i}$ equivalence classes is *represented* somewhere in $C_\mathcal{A}$. In other words, there exists $a_{j_0}, \ldots, a_{j_n} \in A$ such that $[a_j]_{E_i} = [a_{j_i}]_{E_i}$ and

$[a_{j_i}]_{E_i} \in C_{\mathcal{A}} = T_{\mathcal{A}}$ for each $i \in \{0, \ldots, n\}$. Therefore

$$[a_{j_0}]_{E_0} \supseteq [a_{j_1}]_{E_1} \supseteq \cdots \supseteq [a_{j_n}]_{E_n} \supseteq [a_j]_{E_{n+1}},$$

and note that $0 < 1 < \cdots < n+1$. Therefore applying Equation 4.7 for our definition of "$\prec_{\mathcal{T}_{\mathcal{A}}}$",

$$[a_{j_0}]_{E_0} \prec_{\mathcal{T}_{\mathcal{A}}} [a_{j_1}]_{E_1} \prec_{\mathcal{T}_{\mathcal{A}}} \cdots \prec_{\mathcal{T}_{\mathcal{A}}} [a_{j_n}]_{E_n} \prec_{\mathcal{T}_{\mathcal{A}}} [a_j]_{E_{n+1}}$$

Therefore we have put $[a_j]_{E_{n+1}}$ as the greatest element of a path of length $n+2$ under $\prec_{\mathcal{T}_{\mathcal{A}}}$. Since we know that $\mathrm{ht}(\mathcal{T}_{\mathcal{A}}) = n+1$, there can be no paths of length $> n+2$, so there can be no other elements in this path. Furthermore, since $\mathrm{ht}(\mathcal{T}_{\mathcal{A}}) = n+1$, the only way a path like this can exist is if $\mathrm{level}_{\mathcal{T}_{\mathcal{A}}}([a_{j_0}]_{E_0}) = 0$, $\mathrm{level}_{\mathcal{T}_{\mathcal{A}}}([a_{j_1}]_{E_1}) = 1$, $\ldots$, $\mathrm{level}_{\mathcal{T}_{\mathcal{A}}}([a_{j_n}]_{E_n}) = n$, and $\mathrm{level}_{\mathcal{T}_{\mathcal{A}}}([a_j]_{E_{n+1}}) = n+1$. Since a_j was chosen arbitrarily, we can do this for all of $A = \{a_0 < a_1 < \cdots\}$ and in this way we have represented every equivalence class in $C_{\mathcal{A}}$ via this method. Since $C_{\mathcal{A}} = T_{\mathcal{A}}$, we get that for any $[a_k]_{E_i} \in T_{\mathcal{A}}$, $\mathrm{level}_{\mathcal{T}_{\mathcal{A}}}([a_k]_{E_i}) = i$. Furthermore, since for each $j \in \mathbb{N}$, $[a_j]_{E_{n+1}} \in T_{\mathcal{A}}$ and $\mathrm{level}_{\mathcal{T}_{\mathcal{A}}}([a_j]_{E_{n+1}}) = n+1$, this yields that $\mathcal{T}_{\mathcal{A}}$ has infinitely many nodes at level $n+1$.

Now, for any $[a_k]_{E_\ell} \in T_{\mathcal{A}}$, we know that $[a_k]_{E_{n+1}} \in T_{\mathcal{A}}$, and we can create such a path of $(n+2)$-many nodes following the above method:

$$[a_{k_0}]_{E_0} \prec_{\mathcal{T}_{\mathcal{A}}} [a_{k_1}]_{E_1} \prec_{\mathcal{T}_{\mathcal{A}}} \cdots \prec_{\mathcal{T}_{\mathcal{A}}} [a_{k_{\ell-1}}]_{E_{\ell-1}} \prec_{\mathcal{T}_{\mathcal{A}}} [a_k]_{E_\ell} \prec_{\mathcal{T}_{\mathcal{A}}} \cdots \prec_{\mathcal{T}_{\mathcal{A}}} [a_k]_{E_{n+1}}$$

Since we can do this for any node in $T_{\mathcal{A}}$, this means that every node of the tree $T_{\mathcal{A}}$ is on a path of exactly length $n+2$. This means that all paths through $\mathcal{T}_{\mathcal{A}}$ are of the same length, namely length n+2, and hence $\mathcal{T}_{\mathcal{A}}$ is a full tree.

□

Now, let $\mathcal{T}_{\mathcal{A}} = (T_{\mathcal{A}}, \prec_{T_{\mathcal{A}}})$ be the tree as built in Lemma 4.28. We now show that, as built, this tree is indeed computable relative to $\mathcal{A}$.

Lemma 4.30. *$\mathcal{T}_{\mathcal{A}} = (T_{\mathcal{A}}, \prec_{T_{\mathcal{A}}})$ is $\mathcal{A}$-computable.*

Proof. To show that $\mathcal{T}_{\mathcal{A}}$ is $\mathcal{A}$-computable, we need to show that both $T_{\mathcal{A}}$ and $\prec_{T_{\mathcal{A}}}$ are $\mathcal{A}$-computable. Beginning first with $T_{\mathcal{A}}$, we recall that as built, $T_{\mathcal{A}}$"="$C_{\mathcal{A}}$. We already proved in Lemma 4.1 that $C_{\mathcal{A}}$ is an $\mathcal{A}$-computable set. Therefore, given some node of the form $[a]_{E_i}$ we can $\mathcal{A}$-computably tell whether it is in the set $T_{\mathcal{A}}$.

Now, to show that $\prec_{T_\mathcal{A}}$ is $\mathcal{A}$-computable, we need to show that given two nodes $[a_j]_{E_i}, [a_k]_{E_\ell} \in T_\mathcal{A}$, we can $\mathcal{A}$-computably decide whether $[a_j]_{E_i} \prec_{T_\mathcal{A}} [a_k]_{E_\ell}$. Recalling the definition of $\prec_{T_\mathcal{A}}$ given in Equation 4.7, we describe an $\mathcal{A}$-computable process below.

1. First, we check whether $i < \ell$. This is a computable process.
   - If no, then stop. We know that $[a_j]_{E_i} \not\prec_{T_\mathcal{A}} [a_k]_{E_\ell}$.
   - If yes, then keep going.
2. Then check whether $a_j E_i a_k$. This is an $\mathcal{A}$-computable process since E_i is $\mathcal{A}$-computable for each $i \in \{0, \ldots, n+1\}$.
   - If yes, then stop. By Lemma 4.2 $[a_j]_{E_i} \supseteq [a_k]_{E_\ell}$, and hence $[a_j]_{E_i} \prec_{T_\mathcal{A}} [a_k]_{E_\ell}$.
   - If no, then stop. By Lemma 4.2 $[a_j]_{E_i} \not\supseteq [a_k]_{E_\ell}$, and hence $[a_j]_{E_i} \not\prec_{T_\mathcal{A}} [a_k]_{E_\ell}$.

Therefore $\prec_{T_\mathcal{A}}$ is also $\mathcal{A}$-computable, as desired. Hence the tree $\mathcal{T}_\mathcal{A}$ as built in Lemma 4.28 is an $\mathcal{A}$-computable tree. □

This completes the proof of Theorem 4.27. The following corollary is immediate.

Corollary 4.31. *Given a computable n-nested equivalence structure $\mathcal{A} = (A, E_1, \ldots, E_n)$ with infinite domain $A \subseteq \mathbb{N}$, we can define a computable, 1-1, and onto function $\hat{\hat{h}}$ which takes unique equivalence classes of $\mathcal{A}$ and computably builds a full tree, $\mathcal{T}_\mathcal{A}$, of height $n+1$ in which each node represents an equivalence class of $\mathcal{A}$.*

Theorem 4.32. *Let $\mathcal{T}_\mathcal{A} = (T_\mathcal{A}, \prec_{\mathcal{T}_\mathcal{A}})$ be a full tree of height $n+1$ as built in Theorem 4.27, in which each node represents a unique equivalence class of a nested equivalence structure $\mathcal{A} = (A, E_1, \ldots, E_n)$ with infinite domain $A \subseteq \mathbb{N}$. Let $T \subseteq \mathbb{N}$ be any infinite set. Then we can define a function $\hat{h}_T$ which will $(T \oplus \mathcal{A})$-computably build a tree, $\mathcal{T}_{\mathcal{A}_T} = (T_{\mathcal{A}_T}, \prec_{\mathcal{T}_{\mathcal{A}_T}})$, which is isomorphic to $\mathcal{T}_\mathcal{A}$ and which is itself $(T \oplus \mathcal{A})$-computable.*

Proof. The proof is contained in the following lemmas: Lemma 4.33, Lemma 4.34, Lemma 4.35, Lemma 4.36, and Lemma 4.38. □

Lemma 4.33. *Let $\mathcal{T}_\mathcal{A} = (T_\mathcal{A}, \prec_{\mathcal{T}_\mathcal{A}})$ be a full tree of height $n+1$ as built in Theorem 4.27 and let $T \subseteq \mathbb{N}$ be any infinite set. Then we can define a function $\hat{h}_T : T_\mathcal{A} \to T$ which transforms equivalence classes into to natural numbers and is $(T \oplus \mathcal{A})$-computable.*

Proof. We let $\mathcal{T}_{\mathcal{A}} = (T_{\mathcal{A}}, \prec_{\mathcal{T}_{\mathcal{A}}})$ be a full tree of height $n+1$ as built in Theorem 4.27 and let $T \subseteq \mathbb{N}$ be any infinite set. Fix some enumeration of $T = \{t_0 < t_1 < t_2 < \cdots\}$. Additionally, fix some $\mathcal{A}$-computable enumeration of equivalence classes of $\mathcal{A}$ without repetition, $C_{\mathcal{A}} = \{[c_0]_{E_{i_0}}, [c_1]_{E_{i_1}}, [c_2]_{E_{i_2}}, \ldots\}$ with $c_j \in A$, and $i_j \in \{0, \ldots, n+1\}$. Since we built $\mathcal{T}_{\mathcal{A}} = (T_{\mathcal{A}}, \prec_{\mathcal{T}_{\mathcal{A}}})$ such that $C_{\mathcal{A}} = T_{\mathcal{A}}$, this means that we have some $\mathcal{A}$-computable enumeration of nodes of $\mathcal{T}_{\mathcal{A}}$ in which $[c_j]_{E_{i_j}}$ is the jth node in this enumeration (because it is the jth equivalence class in the enumeration of $C_{\mathcal{A}}$). Now define $\hat{h}_T : T_{\mathcal{A}} \to T$ as follows:

$$\hat{h}_T([c_j]_{E_{i_j}}) \stackrel{\text{defn}}{=} t_j \tag{4.9}$$

To show that $\hat{h}_T$ is $(T \oplus \mathcal{A})$-computable, given some input $[a]_{E_i} \in T_{\mathcal{A}}$ we need to show that we can compute $\hat{h}_T([a]_{E_i})$. To do this, we simply $\mathcal{A}$-computably enumerate $C_{\mathcal{A}}$ until $[a]_{E_i}$ shows up. Since $[a]_{E_i} \in T_{\mathcal{A}}$, we are guaranteed this happens at some finite point. Say, $[a]_{E_i}$ shows up as the jth equivalence class enumerated. Then, we T-computably enumerate T in order until we have enumerated the jth element of $T = \{t_0 < t_1 < \cdots\}$. Then we set $\hat{h}([a]_{E_i}) = j$ and halt. □

Lemma 4.34. *Let $\mathcal{T}_{\mathcal{A}} = (T_{\mathcal{A}}, \prec_{\mathcal{T}_{\mathcal{A}}})$ be a full tree of height $n+1$ as built in Theorem 4.27 and let $T \subseteq \mathbb{N}$ be any infinite set. Then the function $\hat{h}_T$ defined in Lemma 4.33 is both 1-1 and onto.*

Proof. To show $\hat{h}_T$ is 1-1, we let $x, y \in T_{\mathcal{A}}$ such that $\hat{h}_T(x) = \hat{h}_T(y)$. Since $x, y \in T_{\mathcal{A}}$, they must be of the form $x = [c_j]_{E_{i_j}}$, $y = [c_k]_{E_{i_k}}$ for some $j, k \in \mathbb{N}$, $i_j, i_k \in \{0, \ldots, n+1\}$.

$$\hat{h}_T(x) = \hat{h}_T([c_j]_{E_{i_j}}) = j = \hat{h}_T([c_k]_{E_{i_k}}) = \hat{h}_T(y) = k$$

Therefore $j = k$, hence $[c_j]_{E_{i_j}} = [c_k]_{E_{i_k}}$ and $x = y$.

To show that $\hat{h}_T$ is onto, we let $t \in T$. Then $t = t_j$ for some jth element of T in the T-computable enumeration $T = \{t_0 < t_1 < t_2 < \cdots\}$. Now, we let $[c_j]_{E_{i_j}}$ represent the jth element in the $\mathcal{A}$-computable enumeration of $C_{\mathcal{A}}$. Then $\hat{h}_T([c_j]_{E_{i_j}}) = t_j = t$, and there exists some element of $T_{\mathcal{A}}$, namely $[c_j]_{E_{i_j}}$, such that applying $\hat{h}_T$ to it yields t. □

Since $\hat{h}_T$ simply relabels the nodes of $\mathcal{T}_{\mathcal{A}}$ in a unique, 1-1 and onto manner, we can define a tree, $\mathcal{T}_{\mathcal{A}_T} = (T_{\mathcal{A}_T}, \prec_{T_{\mathcal{A}_T}})$. Now, we let $T_{\mathcal{A}_T} = T$ for our given infinite set $T \subseteq \mathbb{N}$, and we let $\prec_{T_{\mathcal{A}_T}}$ be inherited directly from $\prec_{\mathcal{T}_{\mathcal{A}}}$. That is,

$$\hat{h}_T([a_j]_{E_i}) \prec_{T_{\mathcal{A}_T}} \hat{h}_T([a_k]_{E_\ell}) \iff [a_j]_{E_i} \prec_{T_{\mathcal{A}}} [a_k]_{E_\ell} \tag{4.10}$$

Equivalently,

$$x \prec_{T_{\mathcal{A}_T}} y \overset{\text{defn}}{\iff} \hat{h}_T^{-1}(x) \prec_{T_{\mathcal{A}}} \hat{h}_T^{-1}(y) \tag{4.11}$$

We now prove what is intuitively fairly obvious — that $\mathcal{T}_{\mathcal{A}_T}$ is indeed a tree and furthermore it is isomorphic to $\mathcal{T}_{\mathcal{A}}$. We also prove that the construction is $(T \oplus \mathcal{A})$-computable.

Lemma 4.35. *Let $\mathcal{A} = (A, E_1, \ldots, E_n)$ be a nested equivalence structure and let $\mathcal{T}_{\mathcal{A}} = (T_{\mathcal{A}}, \prec_{\mathcal{T}_{\mathcal{A}}})$ be its corresponding full tree of height $n+1$ as built in Theorem 4.27. Let $T \subseteq \mathbb{N}$ be any infinite set, let $\hat{h}_T$ be as defined in Lemma 4.33, and let $\prec_{T_{\mathcal{A}_T}}$ be as defined in Equation 4.10. Then $\mathcal{T}_{\mathcal{A}_T} = (T_{\mathcal{A}_T}, \prec_{T_{\mathcal{A}_T}})$ where $T_{\mathcal{A}_T} = T$ defines a tree.*

Proof. To show that $\mathcal{T}_{\mathcal{A}_T} = (T_{\mathcal{A}_T}, \prec_{T_{\mathcal{A}_T}})$ is a tree, we must show that (1) $\prec_{T_{\mathcal{A}_T}}$ is a strict partial order on $T_{\mathcal{A}_T} = T$, and that (2) $\forall x \in T_{\mathcal{A}_T}$, the set of predecessors of x in $T_{\mathcal{A}_T}$ is well-ordered. We begin with (1) and let $t_j, t_k, t_\ell \in T_{\mathcal{A}_T} = T$, and let $[c_j]_{E_{i_j}}$, $[c_k]_{E_{i_k}}$, $[c_\ell]_{E_{i_\ell}}$ be the syntactic equivalents in $T_{\mathcal{A}}$ of, respectively, the jth, kth, and ℓth equivalence classes in our enumeration of $C_{\mathcal{A}}$. Then, by the way we defined $\hat{h}_T$, we know that $\hat{h}_T([c_j]_{E_{i_j}}) = t_j$, $\hat{h}_T([c_k]_{E_{i_k}}) = t_k$, and $\hat{h}_T([c_\ell]_{E_{i_\ell}}) = t_\ell$.

- Irreflexive:

$$\begin{aligned} t_j \prec_{\mathcal{T}_{\mathcal{A}_T}} t_j &\iff [c_j]_{E_{i_j}} \prec_{\mathcal{T}_{\mathcal{A}}} [c_j]_{E_{i_j}} \\ &\iff [c_j]_{E_{i_j}} \supseteq [c_j]_{E_{i_j}} \text{ and } i_j < i_j \end{aligned}$$

  Since we know that for $i_j \in \{0, \ldots, n+1\}$, $i_j \not< i_j$, this means that $t_j \not\prec_{\mathcal{T}_{\mathcal{A}_T}} t_j$, as desired.

- Transitive:

$$\begin{aligned} t_j \prec_{\mathcal{T}_{\mathcal{A}_T}} t_k \ \wedge \ t_k \prec_{\mathcal{T}_{\mathcal{A}_T}} t_\ell &\iff [c_j]_{E_{i_j}} \prec_{\mathcal{T}_{\mathcal{A}}} [c_k]_{E_{i_k}} \ \wedge \ [c_k]_{E_{i_k}} \prec_{\mathcal{T}_{\mathcal{A}}} [c_\ell]_{E_{i_\ell}} \\ &\implies [c_j]_{E_{i_j}} \prec_{\mathcal{T}_{\mathcal{A}}} [c_\ell]_{E_{i_\ell}} \text{ (since } \prec_{\mathcal{T}_{\mathcal{A}}} \text{ transitive)} \\ &\iff t_j \prec_{\mathcal{T}_{\mathcal{A}_T}} t_\ell \end{aligned}$$

- Assymmetric:

$$\begin{aligned} t_j \prec_{\mathcal{T}_{\mathcal{A}_T}} t_k &\iff [c_j]_{E_{i_j}} \prec_{\mathcal{T}_{\mathcal{A}}} [c_k]_{E_{i_k}} \\ &\implies [c_k]_{E_{i_k}} \not\prec_{\mathcal{T}_{\mathcal{A}}} [c_j]_{E_{i_j}} \text{ (since } \prec_{\mathcal{T}_{\mathcal{A}}} \text{ assymetric)} \\ &\iff t_k \not\prec_{\mathcal{T}_{\mathcal{A}_T}} t_j \end{aligned}$$

Now we show (2) that predecessors are well-ordered under $\prec_{\mathcal{T}_{\mathcal{A}_T}}$. Again let $t_j \in T_{\mathcal{A}_T} = T$, and let $[c_j]_{E_{i_j}}$ be the syntactic equivalent in $T_{\mathcal{A}}$ of the jth equivalence class in our enumeration of $C_{\mathcal{A}}$. Now, let $\{t_{p_1}, \ldots, t_{p_r}\}$ be predecessors of t_j under $\prec_{\mathcal{T}_{\mathcal{A}_T}}$ and let $[c_{p_1}]_{E_{i_{p_1}}}$, …, $[c_{p_r}]_{E_{i_{p_r}}}$ be the syntactic equivalents, respectively, in $T_{\mathcal{A}}$ of the p_1th, …, p_rth equivalence classes in our enumeration of $C_{\mathcal{A}}$. By the way we defined $\hat{h}_T$ we know that

$$\hat{h}_T([c_j]_{E_{i_j}}) = t_j$$

$$\hat{h}_T([c_{p_1}]_{E_{i_{p_1}}}) = t_{p_1}$$

$$\vdots$$

$$\hat{h}_T([c_{p_r}]_{E_{i_{p_r}}}) = t_{p_r}$$

Additionally, because $\prec_{\mathcal{T}_{\mathcal{A}_T}}$ is inherited directly from $\prec_{\mathcal{T}_{\mathcal{A}}}$ as in Equation 4.10, we know that $\{[c_{p_1}]_{E_{i_{p_1}}}, \ldots, [c_{p_r}]_{E_{i_{p_r}}}\}$ are all predecessors of $[c_j]_{E_{i_j}}$ under $\prec_{\mathcal{T}_{\mathcal{A}}}$. Therefore this set is well-ordered under $\prec_{\mathcal{T}_{\mathcal{A}}}$ since we already showed that $\mathcal{T}_{\mathcal{A}}$ is a tree; let $[c_{p_0}]_{E_{i_{p_0}}}$ be its least element under $\prec_{\mathcal{T}_{\mathcal{A}}}$. This means that for all $k \in \{1, \ldots, r\}$,

$$\begin{aligned}
[c_{p_0}]_{E_{i_{p_0}}} \preceq_{\mathcal{T}_{\mathcal{A}}} [c_{p_k}]_{E_{i_{p_k}}} &\iff \hat{h}_T([c_{p_0}]_{E_{i_{p_0}}}) \preceq_{\mathcal{T}_{\mathcal{A}}} \hat{h}_T([c_{p_k}]_{E_{i_{p_k}}}) \\
&\qquad \text{(by Equation 4.10)} \\
&\iff t_{p_0} \preceq_{\mathcal{T}_{\mathcal{A}_T}} t_{p_k}
\end{aligned}$$

Since this happens for each $k \in \{1, \ldots, r\}$, therefore t_{p_0} is the least element of $\{t_{p_1}, \ldots, t_{p_r}\}$. Since t_j and its predecessors were chosen arbitrarily, hence any set of predecessors has a least element and is therefore well-ordered under $\prec_{\mathcal{T}_{\mathcal{A}_T}}$, as desired. □

Lemma 4.36. *Let $\mathcal{T}_{\mathcal{A}} = (T_{\mathcal{A}}, \prec_{\mathcal{T}_{\mathcal{A}}})$ be a full tree of height $n+1$ as built in Theorem 4.27 and let $T \subseteq \mathbb{N}$ be any infinite set. Then the function $\hat{h}_T : T_{\mathcal{A}} \to T_{\mathcal{A}_T}$ defined in Lemma 4.33 is a $(T \oplus \mathcal{A})$-computable isomorphism.*

Proof. We already showed that as defined, $\hat{h}_T$ is 1-1 and onto and $(T \oplus \mathcal{A})$-computable. Therefore, we need now only show that order is preserved under $\hat{h}_T$. In other words, we need to show that:

$$x \prec_{\mathcal{T}_{\mathcal{A}}} y \iff \hat{h}_T(x) \prec_{\mathcal{T}_{\mathcal{A}_T}} \hat{h}_T(y)$$

First we note that if $x, y \in T_{\mathcal{A}}$ this means that $x = [c_j]_{E_{i_j}}$ and $y = [c_k]_{E_{i_k}}$ for some $c_j, c_k \in A$ and $i_j, i_k \in \{0, \ldots, n+1\}$, where $[c_j]_{E_{i_j}}, [c_k]_{E_{i_k}}$ are the jth and kth equivalence classes in our enumeration of $C_{\mathcal{A}}$. We also note that we took preservation of order as our definition of $\prec_{\mathcal{T}_{\mathcal{A}_T}}$ in Equation 4.10, and we have:

$$\hat{h}_T([c_j]_{E_{i_j}}) \prec_{\mathcal{T}_{\mathcal{A}_T}} \hat{h}_T([c_k]_{E_{i_k}}) \iff [c_j]_{E_{i_j}} \prec_{\mathcal{T}_{\mathcal{A}}} [c_k]_{E_{i_k}}$$

Therefore $\hat{h}_T$ is a $(T \oplus \mathcal{A})$-computable isomorphism. □

Now that we have confirmed that $\mathcal{T}_{\mathcal{A}_T} = (T_{\mathcal{A}_T}, \prec_{T_{\mathcal{A}_T}})$ is indeed a tree, and furthermore it is isomorphic to $\mathcal{T}_{\mathcal{A}}$, the following properties follow immediately from Corollary 4.29.

Corollary 4.37. *The tree $\mathcal{T}_{\mathcal{A}_T} = (T_{\mathcal{A}_T}, \prec_{T_{\mathcal{A}_T}})$ as built in Lemma 4.35 has the following properties:*

1. *$ht(\mathcal{T}_{\mathcal{A}_T}) = n + 1$*
2. *$level_{\mathcal{T}_{\mathcal{A}_T}}(t_j) = i$, where $[c_j]_{E_i}$ is the jth equivalence class in our enumeration of $C_{\mathcal{A}}$*
3. *$\mathcal{T}_{\mathcal{A}_T}$ has infinitely many nodes at level $n + 1$*
4. *$\mathcal{T}_{\mathcal{A}_T}$ is a full tree*

Lemma 4.38. *The tree $\mathcal{T}_{\mathcal{A}_T} = (T_{\mathcal{A}_T}, \prec_{T_{\mathcal{A}_T}})$ is $(T \oplus \mathcal{A})$-computable.*

Proof. We first need to show that $T_{\mathcal{A}_T}$ is $(T \oplus \mathcal{A})$-computable. But we built $T_{\mathcal{A}_T} = T$, so this is clearly a T-computable set.

We now show that $\prec_{T_{\mathcal{A}_T}}$ is $(T \oplus \mathcal{A})$-computable. Given $x, y \in T_{\mathcal{A}_T}$, below is a process to determine whether $x \prec_{T_{\mathcal{A}_T}} y$.

1. Given $x, y \in T_{\mathcal{A}_T}$ use a T-oracle to start enumerating elements of T in order: $t_0 < t_1 < \cdots$ until x and y have both been enumerated. Determine k, ℓ such that $t_k = x$ and $t_\ell = y$.
2. Then, using an $\mathcal{A}$-oracle, enumerate equivalence classes of $C_{\mathcal{A}}$, until both the kth and the ℓth equivalence classes have been enumerated, call them $[c_k]_{E_i}$ and $[c_\ell]_{E_j}$ respectively.
3. By Lemma 4.33 we know that $\hat{h}_T$ is $(T \oplus \mathcal{A})$-computable. So calculate $\hat{h}_T$ on inputs $[c_k]_{E_i}$ and $[c_\ell]_{E_j}$. This will yield $\hat{h}_T([c_k]_{E_i}) = t_k = x$ and $\hat{h}_T([c_\ell]_{E_j}) = t_\ell = y$.
4. Now, by Lemma 4.30 we know that $\mathcal{T}_{\mathcal{A}}$ is an $\mathcal{A}$-computable tree. Therefore, $\mathcal{A}$-computably determine whether $[c_k]_{E_i} \prec_{\mathcal{T}_{\mathcal{A}}} [c_\ell]_{E_j}$.

- If yes, then stop. We know, by the definition given in Equation 4.10, that $[c_k]_{E_i} \prec_{\mathcal{T}_\mathcal{A}} [c_\ell]_{E_j}$ $\iff \hat{h}_T([c_k]_{E_i}) \prec_{\mathcal{T}_{\mathcal{A}_T}} \hat{h}_T([c_\ell]_{E_j})$, and therefore $x \prec_{\mathcal{T}_{\mathcal{A}_T}} y$.
- If no, then stop. We know, by the definition given in Equation 4.10, that $[c_k]_{E_i} \not\prec_{\mathcal{T}_\mathcal{A}} [c_\ell]_{E_j}$ $\iff \hat{h}_T([c_k]_{E_i}) \not\prec_{\mathcal{T}_{\mathcal{A}_T}} \hat{h}_T([c_\ell]_{E_j})$, and therefore $x \not\prec_{\mathcal{T}_{\mathcal{A}_T}} y$.

The above process relies only on oracles for T and $\mathcal{A}$. Therefore $\prec_{T_{\mathcal{A}_T}}$ is $(T \oplus \mathcal{A})$-computable as desired, and $\mathcal{T}_{\mathcal{A}_T}$ is a $(T \oplus \mathcal{A})$-computable tree. □

This completes the proof of Theorem 4.32. Focusing our examination on computable nested equivalence structures, we get the following corollaries.

Corollary 4.39. *Let $\mathcal{T}_\mathcal{A} = (T_\mathcal{A}, \prec_{\mathcal{T}_\mathcal{A}})$ be a full tree of height $n+1$ as built in Theorem 4.27 and Corollary 4.31, in which each node represents a unique equivalence class of a computable nested equivalence structure $\mathcal{A} = (A, E_1, \ldots, E_n)$ with infinite domain $A \subseteq \mathbb{N}$. Let $T \subseteq \mathbb{N}$ be any computable infinite set. Then we can define a function $\hat{h}_T$ which will computably build a tree $\mathcal{T}_{\mathcal{A}_T} = (T_{\mathcal{A}_T}, \prec_{\mathcal{T}_{\mathcal{A}_T}})$ which is isomorphic to $\mathcal{T}_\mathcal{A}$ and which is itself computable.*

Corollary 4.40. *Let $\mathcal{T}_\mathcal{A} = (T_\mathcal{A}, \prec_{\mathcal{T}_\mathcal{A}})$ be a full tree of height $n+1$ as built in Theorem 4.27 and Corollary 4.31, in which each node represents a unique equivalence class of a computable nested equivalence structure $\mathcal{A} = (A, E_1, \ldots, E_n)$ with infinite domain $A \subseteq \mathbb{N}$. Then we can define a function $\hat{h}_\mathbb{N}$ which will computably build a tree, $\mathcal{T}_{\mathcal{A}_\mathbb{N}} = (T_{\mathcal{A}_\mathbb{N}}, \prec_{\mathcal{T}_{\mathcal{A}_\mathbb{N}}})$ which is isomorphic to $\mathcal{T}_\mathcal{A}$ and which is itself computable.*

Now, we can combine Theorem 4.27 with Theorem 4.32 to yield our intended construction of a tree. We can also focus our examination on computable nested equivalence structures and computable sets, resulting in the following immediate corollaries.

Theorem 4.41. *Given an n-nested equivalence structure $\mathcal{A} = (A, E_1, \ldots, E_n)$ with infinite domain $A \subseteq \mathbb{N}$, and given any infinite set $T \subseteq \mathbb{N}$, we can build a full, $(T \oplus \mathcal{A})$-computable tree, $\mathcal{T}_{\mathcal{A}_T} = (T_{\mathcal{A}_T}, \prec_{\mathcal{T}_{\mathcal{A}_T}})$, which has finite height $n+1$ and which converts the equivalence classes of $\mathcal{A}$ into nodes on the tree, and furthermore displays the nesting properties, "$\subseteq$", of $\mathcal{A}$, as branching properties, "$\prec_{\mathcal{T}_{\mathcal{A}_T}}$", on $\mathcal{T}_{\mathcal{A}_T}$.*

Corollary 4.42. *Given a computable n-nested equivalence structure $\mathcal{A} = (A, E_1, \ldots, E_n)$ with infinite domain $A \subseteq \mathbb{N}$, and given any infinite computable set $T \subseteq \mathbb{N}$, we can build a full, computable tree, $\mathcal{T}_{\mathcal{A}_T} = (T_{\mathcal{A}_T}, \prec_{\mathcal{T}_{\mathcal{A}_T}})$, which has finite height $n+1$ and which converts the equivalence classes of $\mathcal{A}$ into nodes on the tree, and furthermore displays the nesting properties, "$\subseteq$", of $\mathcal{A}$, as branching properties, "$\prec_{\mathcal{T}_{\mathcal{A}_T}}$", on $\mathcal{T}_{\mathcal{A}_T}$.*

Corollary 4.43. *Given a computable n-nested equivalence structure $\mathcal{A} = (A, E_1, \ldots, E_n)$ with infinite domain $A \subseteq \mathbb{N}$, we can build a full, computable tree, $\mathcal{T}_{\mathcal{A}_{\mathbb{N}}} = (T_{\mathcal{A}_{\mathbb{N}}}, \prec_{\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}})$, which has finite height $n+1$ and which converts the equivalence classes of $\mathcal{A}$ into nodes on the tree, and furthermore displays the nesting properties, "$\subseteq$", of $\mathcal{A}$, as branching properties, "$\prec_{\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}}$", on $\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}$.*

4.6 Putting It All Together

Taking Theorem 4.24 and Theorem 4.41 together, we can now computably go back and forth between nested equivalence structures and trees. The notation we have used for each part is almost a mirror image of the other. In Theorem 4.24 we began with a tree, $\mathcal{T} = (T, \prec_{\mathcal{T}})$, took a set $A \subseteq \mathbb{N}$ and built an isomorphic tree, $\mathcal{T}_A = (T_A, \prec_{\mathcal{T}_A})$, from it. From there, we then finally built a nested equivalence structure, $\mathcal{A}_{\mathcal{T}_A} = (A, E_1^{\mathcal{A}_{\mathcal{T}_A}}, \ldots, E_n^{\mathcal{A}_{\mathcal{T}_A}})$. Conversely, in Theorem 4.41 we began with a nested equivalence structure $\mathcal{A} = (A, E_1, \ldots, E_n)$, built an intermediate tree $\mathcal{T}_{\mathcal{A}} = (T_{\mathcal{A}} \prec_{\mathcal{T}_{\mathcal{A}}})$ from it, and then finally relabelled nodes so they consisted of natural numbers from some set T, to yield a tree $\mathcal{T}_{\mathcal{A}_T} = (T_{\mathcal{A}_T}, \prec_{T_{\mathcal{A}_T}})$.

If we examine this notation carefully we see that the trees and nested equivalence structures are not entirely mirror images of each other. For instance, $\mathcal{T}_{\mathcal{A}}$ and $\mathcal{T}_A$ do not represent identical objects. The structure of the tree $\mathcal{T}_{\mathcal{A}}$ built from a nested equivalence structure $\mathcal{A}$ depends entirely upon the construction of the nested equivalence structure and its properties. The structure of the tree $\mathcal{T}_A$ built from a tree $\mathcal{T}$ and a set of natural numbers A depends entirely upon the construction and the properties of the given tree, T, and almost not at all on the set of numbers A. The set A only provides a labelling convention and does not contribute any structure to the built tree.

If we examine Figure 4.3 and Figure 4.4 together, and we follow our methods of Theorem 4.24 and Theorem 4.41 together we may not get what we expect. For instance, if we take a tree $\mathcal{T}$ as input, create a nested equivalence structure $\mathcal{A}_{\mathcal{T}_A}$ out of it, and then create a tree $\mathcal{T}_{(\mathcal{A}_{\mathcal{T}_A})_T}$ back out of $\mathcal{A}_{\mathcal{T}_A}$, we will not get our original tree $\mathcal{T}$ back in return. That is, it is not necessarily true that $\mathcal{T} = \mathcal{T}_{(\mathcal{A}_{\mathcal{T}_A})_T}$. Similarly, it is also not necessarily true that $\mathcal{A} = \mathcal{A}_{(\mathcal{T}_{\mathcal{A}_T})_A}$. This remains the case even if we ensure that the domains we feed our algorithms (either A or T) remain exactly the same at each point through the process.

On the surface, this many seem like an unfortunate result. Our efforts have not been in vain, however. While we may not get back the *exact* same structure in return, we will indeed get back a computably isomorphic one. That is, taking Theorem 4.24 and Theorem 4.41 together we do have that $\mathcal{T} \simeq_{\mathcal{T} \oplus A} \mathcal{T}_{(\mathcal{A}_{\mathcal{T}_A})_T}$ and $\mathcal{A} \simeq_{T \oplus \mathcal{A}} \mathcal{A}_{(\mathcal{T}_{\mathcal{A}_T})_A}$, and in particular for computable sets and

structures, $\mathcal{T} \simeq_c \mathcal{T}_{(\mathcal{A}_{\mathcal{T}_A})_T}$ and $\mathcal{A} \simeq_c \mathcal{A}_{(\mathcal{T}_{\mathcal{A}_T})_A}$. (We formally prove these results in Theorem 5.10 and Corollary 5.16.) Therefore, applying Theorem 4.24 and Theorem 4.41, we not only preserve the structure of the given object (nested equivalence structure or tree), but we do so computably, and hence preserve their computability-theoretic properties when going back and forth between the two. We further explore this concept in the following chapter.

Chapter 5

Category-Theoretic Notions of Trees and Nested Equivalence Structures

In the previous chapter, we formalized algorithms to go back and forth between nested equivalence structures and trees of finite height. These algorithms worked in the most generic sense. They took any domain, A or T, as input, and output a nested equivalence structure or a tree with that given domain. Additionally, these algorithms were computable relative to the various inputs (either a tree $\mathcal{T}$ and domain A, or a nested equivalence structure $\mathcal{A}$ and a domain T). We now turn our attention to the language of category theory to further solidify these notions of going between nested equivalence structures and trees. By using the concepts and tools of categories and functors, we can in this way take our generic algorithms of the previous section and fix a specific method to turn a given tree into a nested equivalence structure and vice versa. Once fixing these specific methods, we can then show that many nice properties hold. In particular, the structures we build will maintain isomorphisms and Turing computability.

We build off of existing work which uses similar tools to transfer computability-theoretic results about one type of structure to another. For instance, in [28] Hirschfeldt, Khoussainov, Shore, and Slinko took a certain class of countable structures and turned them into a certain class of graphs, such that isomorphisms between the initial two structures were maintained as isomorphisms between the graphs. In [35] Miller, Park, Poonen, Schoutens, and Shlapentokh then expanded upon this work

and built a functor, then, from a certain class of graphs to a certain class of fields and similarly extended the computability-theoretic results. In [38] Ocasio and Knight transferred results from countable real closed fields to countable linear orders. And in [26], Harrison-Trainor, Melnikov, Miller, and Montalbán considered effective interpretability of a structure and examined this notion within the context of computable functors. We now continue here and use category-theoretic notions to examine nested equivalence structures and trees of finite height.

Before proceeding, we note that the methods we will use in this section would work equally well for trees and nested equivalence structures with finite domains. Because finite height trees with finite domain are necessarily finite, and so too are finitely nested equivalence structures with finite domains, we wish to concentrate here on the more general case — where our tree or nested equivalence structure has countably infinite domain.

5.1 The Categories: FFT and NEquiv

To use the tools of category theory to discuss trees and nested equivalence structures, we first begin by defining appropriate categories for each. Theorem 4.24 required that in order to build a finitely nested equivalence structure, we first must begin with a full finite height tree with at least 3 levels (height ≥ 2). Furthermore, Corollary 4.37 confirmed that if we began with a finitely nested equivalence structure with at least one equivalence relation, the tree that we built would be full, and of finite height ≥ 2.

We formally define **FFT** to be the category of full finite height trees with infinite domain and height ≥ 2, as follows. An *object* in **FFT** consists of a tree, $\mathcal{T} = (T, \prec_{\mathcal{T}})$, which satisfies all of the following conditions:

- $T \subseteq \mathbb{N}$ is infinite
- $\prec_{\mathcal{T}}$ is a strict partial order on T under which sets of predecessors are well-ordered
- $2 \leq \mathrm{ht}(\mathcal{T}) < \omega$
- All paths in T are of the same length, that is $\mathcal{T}$ is full

A *morphism* in **FFT** consists of an isomorphism between trees. That is, for full finite height trees $\mathcal{T}, \mathcal{S} \in \mathbf{FFT}$, $f : \mathcal{T} \to \mathcal{S}$ is a morphism in **FFT** iff $f : T \to S$ is a 1-1 and onto function which preserves order:

$$x \prec_{\mathcal{T}} y \iff f(x) \prec_{\mathcal{S}} f(y)$$

To confirm that **FFT** is indeed a category in the formal sense, we must check that composition of isomorphisms between trees is still an isomorphism, composition is associative, and that unity for morphisms is satisfied. We do that now.

Theorem 5.1. ***FFT*** *is a category.*

Proof. Let $\mathcal{T}, \mathcal{S}, \mathcal{U}, \mathcal{V} \in \mathbf{FFT}$ be trees, and let $f : \mathcal{T} \to \mathcal{S}$, $g : \mathcal{S} \to \mathcal{U}$, and $h : \mathcal{U} \to \mathcal{V}$ be isomorphisms on these trees. We begin by checking that composition of isomorphisms between full, finite height trees is still an isomorphism between full finite height trees. This is a straightforward result. We know that $(g \circ f) : \mathcal{T} \to \mathcal{U}$ is both 1-1 and onto since both g and f are. Therefore $(g \circ f)(\mathcal{T}) = \mathcal{U}$, and we already know that $\mathcal{U} \in \mathbf{FFT}$. Additionally $(g \circ f)$ preserves order as shown below:

$$\begin{aligned} x \prec_{\mathcal{T}} y &\iff f(x) \prec_{\mathcal{S}} f(y) && \text{(since } f \text{ preserves order)} \\ &\iff g(f(x)) \prec_{\mathcal{U}} g(f(y)) && \text{(since } g \text{ preserves order)} \\ &\iff (g \circ f)(x) \prec_{\mathcal{U}} (g \circ f)(y) \end{aligned}$$

Now, we check that composition is indeed associative. We need to show that $h \circ (g \circ f) = (h \circ g) \circ f$. Let $x \in \mathcal{T}$. We do so below.

$$\big(h \circ (g \circ f)\big)(x) = h \circ \big((g \circ f)(x)\big) = h \circ \big(g(f(x))\big) = h\big(g(f(x))\big)$$

$$\big((h \circ g) \circ f\big)(x) = (h \circ g)(f(x)) = h\big(g(f(x))\big)$$

Finally, we check that unity for morphisms is satisfied. In particular, the identity function, Id, will do just that. Clearly Id is 1-1 and onto. Additionally Id preserves order. To see this, consider $\mathrm{Id} : \mathcal{T} \to \mathcal{T}$. Then,

$$x \prec_{\mathcal{T}} y \iff \mathrm{Id}(x) = x \prec_{\mathcal{T}} y = \mathrm{Id}(y) \iff \mathrm{Id}(x) \prec_{\mathcal{T}} \mathrm{Id}(y)$$

Therefore Id is indeed an isomorphism. Furthermore, Id satisfies unity of morphisms. Taking f as above we have,

$$(\mathrm{Id}_{\mathcal{S}} \circ f)(x) = \mathrm{Id}_{\mathcal{S}}\big(f(x)\big) = f(x)$$

$$(f \circ \mathrm{Id}_{\mathcal{T}})(x) = f\big(\mathrm{Id}_{\mathcal{T}}(x)\big) = f\big(x\big)$$

Therefore **FFT** is a category. □

We formally define **NEquiv** to be the category of finitely nested equivalence structures with infinite domain, as follows. An *object* in **NEquiv** consists of a nested equivalence structure $\mathcal{A} = (A, E_1, \ldots, E_n)$ which satisfies all of the following conditions:

- $A \subseteq \mathbb{N}$ is infinite
- $E_1, \ldots, E_n$ are equivalence relations on A
- $0 < n < \omega$
- $\forall a \in A$, $[a]_{E_1} \supseteq [a]_{E_2} \supseteq \cdots \supseteq [a]_{E_n}$

A *morphism* in **NEquiv** consists of an isomorphism between nested equivalence structures. That is, for nested equivalence structures $\mathcal{A} = (A, E_1, \ldots, E_n)$, and $\mathcal{B} = (B, R_1, \ldots, R_n)$, each morphism $f : \mathcal{A} \to \mathcal{B}$ consists of a 1-1 and onto function $f : A \to B$ which preserves equivalence relations:

$$\forall i \in \{1, \ldots, n\},\ xE_iy \iff f(x)R_if(y)$$

Note that if equivalence relations are preserved, then so is their nesting. Let $a \in A$. Then we have,

$$x \in [a]_{E_i} \iff xE_ia \iff f(x)R_if(a) \iff f(x) \in [f(a)]_{R_i}$$

$$\begin{aligned} \therefore\ [a]_{E_{i+1}} \subseteq [a]_{E_i} &\iff \Big(x \in [a]_{E_{i+1}} \implies x \in [a]_{E_i}\Big) \\ &\iff \Big(f(x) \in [f(a)]_{R_{i+1}} \implies f(x) \in [f(a)]_{R_i}\Big) \\ &\iff [f(a)]_{R_{i+1}} \subseteq [f(a)]_{R_i} \end{aligned}$$

Therefore, when building isomorphisms between nested equivalence structures, it is only necessary to ensure that each of the equivalence relations is preserved under the isomorphism; the desired nesting will then follow as an immediate consequence.

To confirm that **NEquiv** is indeed a category in the formal sense, we now must check that composition of isomorphisms between nested equivalence structures is still an isomorphims, that composition is associative, and that unity for morphisms is satisfied. These results are all straightforward.

Theorem 5.2. ***NEquiv*** *is a category.*

Proof. Let $\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{D} \in \mathbf{NEquiv}$ be n-nested equivalence structures with equivalence relations E_i, R_i, P_i, and Q_i for $i = 1, \ldots, n$ respectively. Let $f : \mathcal{A} \to \mathcal{B}$, $g : \mathcal{B} \to \mathcal{C}$, and $h : \mathcal{C} \to \mathcal{D}$. We first check that composition of isomorphisms between nested equivalence structures yields an isomorphism between nested equivalence structures. First, both g and g are 1-1 and onto since they are isomorphisms, therefore $(g \circ f) : \mathcal{A} \to \mathcal{C}$ is 1-1 and onto. Furthermore, $(g \circ f)(\mathcal{A}) = \mathcal{C} \in \mathbf{NEquiv}$. Now to see that $(g \circ f)$ preserves equivalence relations let $i \in \{1, \ldots, n\}$,

$$\begin{aligned} xE_iy &\iff f(x)R_if(y) && \text{(since } f \text{ preserves equiv. rltns)} \\ &\iff g(f(x))P_ig(f(y)) && \text{(since } g \text{ preserves equiv. rltns)} \\ &\iff (g \circ f)(x)P_i(g \circ f)(y) \end{aligned}$$

Therefore composition of isomorphisms between n-nested equivalence structures yields an isomorphism between n-nested equivalence structures.

Next, we check that composition of isomorphisms is associative. First we note that:

$$\big(h \circ (g \circ f)\big)(x) = h \circ \big((g \circ f)(x)\big) = h \circ \big(g(f(x))\big) = h\big(g(f(x))\big)$$

And next we see that:

$$\big((h \circ g) \circ f\big)(x) = (h \circ g)(f(x)) = h\big(g(f(x))\big)$$

Therefore $h \circ (g \circ f) = (h \circ g) \circ f$, and hence composition of isomorphisms of nested equivalence structures is associative.

Finally, we need to check that there is a unity morphism in **NEquiv**. Note that the identitiy function, Id, will again do just that. Clearly Id is 1-1 and onto. Additionally, Id preserves equivalence relations as noted below for $i \in \{1, \ldots, n\}$

$$xE_iy \iff \mathrm{Id}(x) = x\, E_i\, y = \mathrm{Id}(y) \iff \mathrm{Id}(x)E_i\mathrm{Id}(y)$$

Therefore Id is an isomorphism between nested equivalence structures. Additionally, Id satisfies the necessary unity properties.

$$(\mathrm{Id}_{\mathcal{S}} \circ f)(x) = \mathrm{Id}_{\mathcal{S}}\big(f(x)\big) = f(x)$$

$$(f \circ \mathrm{Id}_{\mathcal{T}})(x) = f\big(\mathrm{Id}_{\mathcal{T}}(x)\big) = f\big(x\big)$$

Therefore **NEquiv** satisfies all the necessary conditions, and **NEquiv** is indeed a category. □

Before moving on, we note the standard abuse of notation here. As a morphism, f takes one object to another in a given category. In both of our categories, **FFT** and **NEquiv**, objects are structures. Therefore, a morphism f takes one structure to another in these categories. As function, however, f takes the domain of one structure to the domain of another structure. Since we are considering morphisms in **FFT** and **NEquiv** to be functions which are isomorphisms on their respective structures, the structure is maintained under this application of the function to the respective domains. Therefore, for our purposes it is appropriate to talk about a "function" f taking one structure to another, or about a "morphism" f being a function on the domains of structure. We will use these concepts almost interchangeably.

5.2 Functor from FFT to NEquiv

Now that we have defined the categories **FFT** and **NEquiv**, we can build functors which fix a specific method of going between trees and nested equivalence structures.

Eventually, we wish for these functors to be full, faithful, essentially onto, and computable relative to the various structures involved. To clarify what we need to show, Figure 5.1 shows the functor $F : \mathbf{FFT} \to \mathbf{NEquiv}$ and describes all the different interim functions that we will use in our various definitions and proofs.

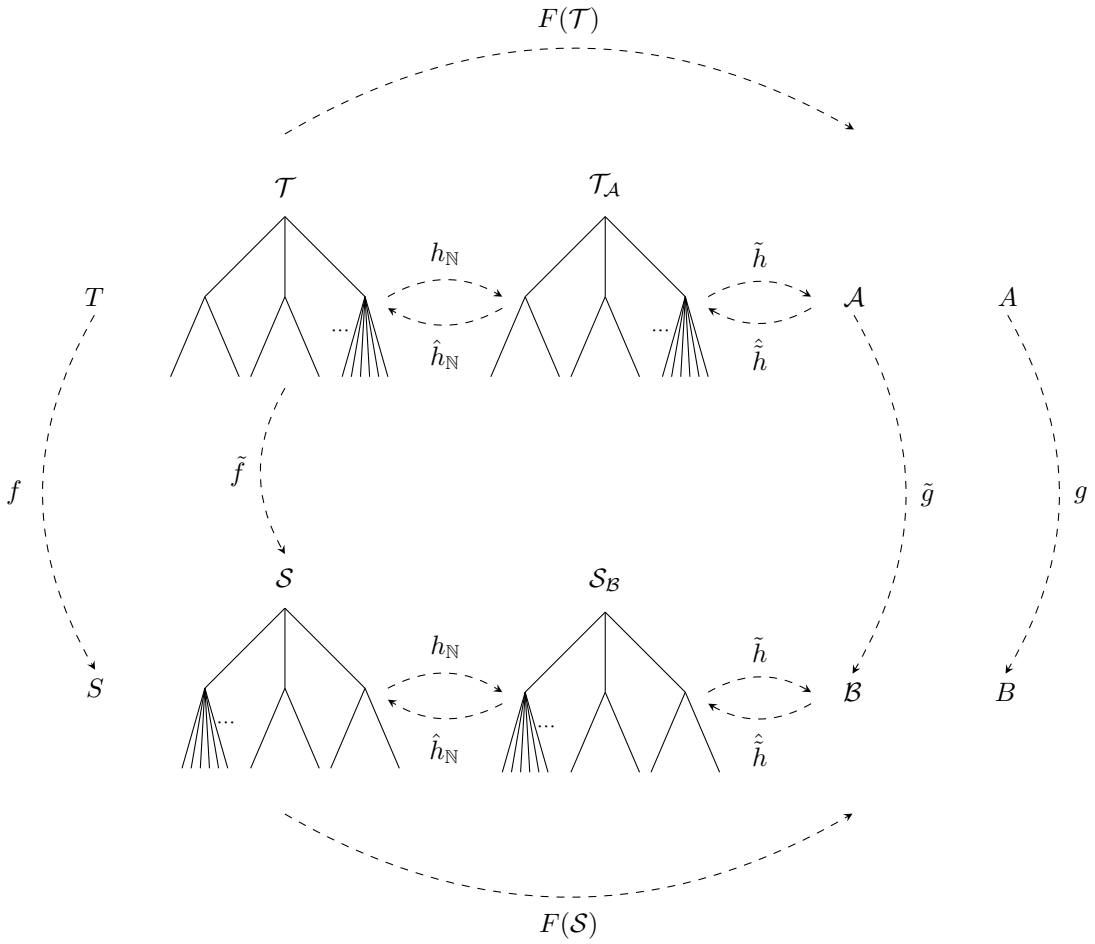


Figure 5.1: Diagram representing the functor, F, from the category **FFT** to the category **NEquiv** and representing all associated sub-mappings in between.

We now define a functor $F : \mathbf{FFT} \to \mathbf{NEquiv}$. Since functors take objects to objects and morphisms to morphisms, there are two parts to this definition. We let $\mathcal{T}, \mathcal{S}, f \in \mathbf{FFT}$, with $\mathcal{T} = (T, \prec_{\mathcal{T}})$, $\mathcal{S} = (S, \prec_{\mathcal{S}})$, and $f : \mathcal{T} \to \mathcal{S}$. Additionally, let $\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}$ and $\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}$ represent the nested equivalence structures built from trees $\mathcal{T}$ and $\mathcal{S}$ and set $\mathbb{N}$ as in Theorem 4.24. As built, $\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}$ and $\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}$ are clearly objects in **NEquiv**. We define F as follows.

- Objects: $F(\mathcal{T}) = \mathcal{A}_{\mathcal{T}_{\mathbb{N}}} = (\mathbb{N}, E_1^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}, \ldots, E_n^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}})$
- Morphisms: $F(f) = g$ where $g : \mathcal{A}_{\mathcal{T}_{\mathbb{N}}} \to \mathcal{A}_{\mathcal{S}_{\mathbb{N}}}$ is defined as follows.

Let $\mathcal{T}, \mathcal{S}$ both have height $n+1$, and let $\{e_{\mathcal{T},0}, e_{\mathcal{T},1}, e_{\mathcal{T},2}, \ldots\}$ and $\{e_{\mathcal{S},0}, e_{\mathcal{S},1}, e_{\mathcal{S},2}, \ldots\}$ represent our $\mathcal{T}$-computable and $\mathcal{S}$-computable enumerations of end nodes of $\mathcal{T}$ and $\mathcal{S}$ respectively as in Lemma 4.8. Note that by the first part of our functor definition, $F(\mathcal{T}) = \mathcal{A}_{\mathcal{T}_{\mathbb{N}}} = (\mathbb{N}, E_1^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}, \ldots, E_n^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}})$ and similarly $F(\mathcal{S}) = \mathcal{A}_{\mathcal{S}_{\mathbb{N}}} = (\mathbb{N}, E_1^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}, \ldots, E_n^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}})$. Therefore g should be an isomorphism between

the two structures, $\mathcal{A}_{\mathcal{T}_\mathbb{N}}$ and $\mathcal{A}_{\mathcal{S}_\mathbb{N}}$. We can now define $g : \mathbb{N} \to \mathbb{N}$ as follows.

$$g(j) = k \overset{\text{defn}}{\iff} f(e_{\mathcal{T},j}) = e_{\mathcal{S},k} \tag{5.1}$$

This definition may seem a little odd. After all, we are relying only upon end-nodes of $\mathcal{T}$ and $\mathcal{S}$ to define an isomorphism from $\mathcal{A}_{\mathcal{T}_\mathbb{N}}$ to $\mathcal{A}_{\mathcal{S}_\mathbb{N}}$. If we recall the construction of $\mathcal{A}_{\mathcal{T}_\mathbb{N}}$, however, the first step involved placing all elements of our given set (in this case, $\mathbb{N}$) onto the terminating level of a tree with nodes that looked like equivalence classes. Indeed, Corollary 4.16 confirmed that each element of our set (again, in this case $\mathbb{N}$) was represented on the terminating level of this tree. We also note that since f is itself an isomorphism, we are guaranteed that all the "middle" levels of the trees $\mathcal{T}$ and $\mathcal{S}$ line up with each other under $\prec_\mathcal{T}$ and $\prec_\mathcal{S}$ appropriately. Hence so, too, will all the "middle" levels of $\mathcal{T}_\mathbb{N}$ and $\mathcal{S}_\mathbb{N}$ align, and subsequently all equivalence classes of $\mathcal{A}_{\mathcal{T}_\mathbb{N}}$ and $\mathcal{A}_{\mathcal{S}_\mathbb{N}}$ once we finish their construction. Though this is not necessarily intuitively clear, we will now confirm that g as defined is indeed an isomorphism.

Lemma 5.3. *The function $g = F(f)$ as defined in Equation 5.1 is an isomorphism.*

Proof. Let $\mathcal{T}, \mathcal{S}$ be full, finite height trees of height $n+1 \geq 2$, and let $\mathcal{A}_{\mathcal{T}_\mathbb{N}} = (\mathbb{N}, E_1^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}, \ldots, E_n^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}})$ and $\mathcal{A}_{\mathcal{S}_\mathbb{N}} = (\mathbb{N}, E_1^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}, \ldots, E_n^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}})$ be nested equivalence structures built from them and the set $\mathbb{N}$ as in Theorem 4.24. To prove that $g : \mathcal{A}_{\mathcal{T}_\mathbb{N}} \to \mathcal{A}_{\mathcal{S}_\mathbb{N}}$ is an isomorphism, we need to prove that it is 1-1, onto, and preserves equivalence relations. Let $\{e_{\mathcal{T},0}, e_{\mathcal{T},1}, e_{\mathcal{T},2}, \ldots\}$ and $\{e_{\mathcal{S},0}, e_{\mathcal{S},1}, e_{\mathcal{S},2}, \ldots\}$ represent our $\mathcal{T}$-computable and $\mathcal{S}$-computable enumerations of end nodes of $\mathcal{T}$ and $\mathcal{S}$ respectively as in Lemma 4.8.

We begin by showing g is 1-1. Let $j, \ell \in \mathbb{N}$ such that $g(j) = g(\ell)$. Then,

$$\begin{aligned} g(j) = k = g(\ell) &\iff f(e_{\mathcal{T},j}) = e_{\mathcal{S},k} = f(e_{\mathcal{T},\ell}) && \text{(by Equation 5.1)} \\ &\implies e_{\mathcal{T},j} = e_{\mathcal{T},\ell} && \text{(since } f \text{ an isomorphism, } \therefore \text{ 1-1)} \end{aligned}$$

Therefore the jth terminating node in our enumeration of terminating nodes of $\mathcal{T}$ is the same as the ℓth terminating node in our enumeration of terminating nodes of $\mathcal{T}$. But our enumeration of end nodes lists each end node exactly once, without repetition. Therefore, we must have that $j = \ell$, and hence $g(j) = g(\ell) \implies j = \ell$ as desired.

To show that g is onto, we let $k \in \mathbb{N}$. Since we are considering trees, $\mathcal{S}$ and $\mathcal{T}$, with countably infinite domains, we know that there are infinitely many nodes at the terminating level of each tree. Therefore our enumeration of end nodes of $\mathcal{S}$ is indeed an infinite set, and hence there is some kth

element in its enumeration, call it $e_{\mathcal{S},k}$. Since f is an isomorphism, and hence 1-1 and onto, there is some corresponding element in $\mathcal{T}$ such that applying f yields $e_{\mathcal{S},k}$. Additionally, since f is an ismorphism, and hence preserves order, this element of $\mathcal{T}$ must itself be an end node of $\mathcal{T}$ since $e_{\mathcal{S},k}$ is an end node of $\mathcal{S}$. In other words, there is some $j \in \mathbb{N}$ such that $e_{\mathcal{T},j}$ is a terminating node of $\mathcal{T}$ and $f(e_{\mathcal{T},j}) = e_{\mathcal{S},k}$. Therefore, by how we defined g in Equation 5.1, $g(j) = k$, and there is an element of $\mathbb{N}$, namely $j \in \mathbb{N}$, such that $g(j) = k$. Therefore g is onto.

Now, to show that g preserves equivalence relations we need to show that given $j, \ell \in \mathbb{N}$, for each $i \in \{1, \ldots, n\}$,

$$jE_i^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}\ell \iff g(j)E_i^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}g(\ell)$$

To show this, we think back to the construction of $\mathcal{A}_{\mathcal{T}_\mathbb{N}}$ and $\mathcal{A}_{\mathcal{S}_\mathbb{N}}$ in Theorem 4.10 and Theorem 4.17. We follow elements of $\mathcal{A}_{\mathcal{T}_\mathbb{N}}$ back through the construction to $\mathcal{T}$, then apply the isomorphism f, and follow them again back the other way through the construction from $\mathcal{S}$ to $\mathcal{A}_{\mathcal{S}_\mathbb{N}}$. The following does just this.

$$\begin{aligned}
jE_i^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}\ell \iff & \text{(by Equation 4.6)} \quad \exists \text{ node } x \text{ at level } i \text{ of } \mathcal{T}_\mathbb{N} \text{ such that:} \\
& x \prec_{\mathcal{T}_\mathbb{N}} [j]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}} \text{ and } x \prec_{\mathcal{T}_\mathbb{N}} [\ell]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}} \\
\iff & \text{(by Equation 4.5)} \quad \exists \text{ node } (h_\mathbb{N}^{\mathcal{T}})^{-1}(x) \text{ at level } i \text{ of } \mathcal{T} \text{ such that:} \\
& (h_\mathbb{N}^{\mathcal{T}})^{-1}(x) \prec_{\mathcal{T}} (h_\mathbb{N}^{\mathcal{T}})^{-1}([j]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}}) \text{ and } (h_\mathbb{N}^{\mathcal{T}})^{-1}(x) \prec_{\mathcal{T}_\mathbb{N}} (h_\mathbb{N}^{\mathcal{T}})^{-1}([\ell]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}}) \\
\iff & \text{(by Lemma 4.11)} \quad \exists \text{ node } y \text{ at level } i \text{ of } \mathcal{T} \text{ such that:} \\
& y \prec_{\mathcal{T}} e_{\mathcal{T},j} \text{ and } y \prec_{\mathcal{T}} e_{\mathcal{T},\ell} \\
\iff & \text{(b/c } f \text{ an isomorphism)} \quad \exists \text{ node } f(y) \text{ at level } i \text{ of } \mathcal{S} \text{ such that:} \\
& f(y) \prec_{\mathcal{S}} f(e_{\mathcal{T},j}) \text{ and } f(y) \prec_{\mathcal{S}} f(e_{\mathcal{T},\ell}) \\
\iff & \exists \text{ node } f(y) \text{ at level } i \text{ of } \mathcal{S} \text{ such that:} \\
& f(y) \prec_{\mathcal{S}} e_{\mathcal{S},k} \text{ and } f(y) \prec_{\mathcal{S}} e_{\mathcal{S},m} \text{ for some } k, m \in \mathbb{N}
\end{aligned}$$

Note in the prior step, because f is an isomorphism and $e_{\mathcal{T},j}$, $e_{\mathcal{T},\ell}$ are terminating nodes of $\mathcal{T}$, this necessarily means that $f(e_{\mathcal{T},j})$ and $f(e_{\mathcal{T},\ell})$ are terminating nodes of $\mathcal{S}$. Therefore the nodes $f(e_{\mathcal{T},j})$ and $f(e_{\mathcal{T},\ell})$ appear somewhere in the enumeration of all terminating nodes of $\mathcal{S}$, call it at the kth and mth place respectively. That is, $f(e_{\mathcal{T},j}) = e_{\mathcal{S},k}$ and $f(e_{\mathcal{T},\ell})e_{\mathcal{S},m}$. Note that it is not necessarily true that $j < \ell \implies k < m$ (or vice versa), but this fact is not needed. Continuing on we have,

$$
\begin{aligned}
&\iff \text{(by Equation 4.4)} \quad \exists \text{ node } h^{\mathcal{S}}_{\mathbb{N}}(f(y)) \text{ at level } i \text{ of } \mathcal{S}_{\mathbb{N}} \text{ such that:} \\
&\quad h^{\mathcal{S}}_{\mathbb{N}}(f(y)) \prec_{\mathcal{S}_{\mathbb{N}}} h^{\mathcal{S}}_{\mathbb{N}}(e_{\mathcal{S},k}) \text{ and } h^{\mathcal{S}}_{\mathbb{N}}(f(y)) \prec_{\mathcal{S}_{\mathbb{N}}} h^{\mathcal{S}}_{\mathbb{N}}(e_{\mathcal{S},m}) \\
&\iff \text{(by Lemma 4.11)} \quad \exists \text{ node } z = h^{\mathcal{S}}_{\mathbb{N}}(f(y)) \text{ at level } i \text{ of } \mathcal{S}_{\mathbb{N}} \text{ such that:} \\
&\quad z \prec_{\mathcal{S}_{\mathbb{N}}} [k]_{E^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}_{n+1}} \text{ and } z \prec_{\mathcal{S}_{\mathbb{N}}} [m]_{E^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}_{n+1}} \\
&\iff \text{(by Equation 4.6)} \\
&\quad k E_i^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}} m \\
&\iff \big(\text{applying our definition of } g \text{ to: } f(e_{\mathcal{T},j}) = e_{\mathcal{S},k} \ \wedge \ f(e_{\mathcal{T},\ell})e_{\mathcal{S},m}\big) \\
&\quad g(j) E_i^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}} g(\ell)
\end{aligned}
$$

Combining this all together yields $jE_i^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}\ell \iff g(j)E_i^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}g(\ell)$. Hence g preserves equivalence relations.

Therefore we have shown that $g : \mathcal{A}_{\mathcal{T}_{\mathbb{N}}} \to \mathcal{A}_{\mathcal{S}_{\mathbb{N}}}$ is 1-1, onto, and preserves equivalence relations. Therefore g is an isomorphism on nested equivalence structures. □

Since g is indeed an isomorphism between objects of **NEquiv**, we can say with certainty that $g \in$ **NEquiv**. Therefore, we have that $F(f : \mathcal{T} \to \mathcal{S}) = g : \mathcal{A}_{\mathcal{T}_{\mathbb{N}}} \to \mathcal{A}_{\mathcal{S}_{\mathbb{N}}} = F(f) : F(\mathcal{T}) \to F(\mathcal{S})$. Now, although $F :$ **FFT** $\to$ **NEquiv**, appears to correctly define a functor which takes objects to objects and morphisms to morphisms, we still need to check that a few properties are satisfied before we can confirm that F does indeed define a functor. Namely, we need to check that identity and composition are both preserved. We do that now.

Lemma 5.4. *F preserves identity.*

Proof. We let Id represent the identity function. As discussed in Section 5.1, the identity function serves as the unity morphism for both **FFT** and **NEquiv**. Let $\mathcal{T} \in$ **FFT**, $\mathrm{Id}_{\mathcal{T}} : \mathcal{T} \to \mathcal{T}$. Then $F(\mathrm{Id}_{\mathcal{T}}) = g$ where $g : F(\mathcal{T}) \to F(\mathcal{T})$ and following Equation 5.1, g is defined as follows for $j, k \in \mathbb{N}$,

$$g(j) = k \overset{\text{defn}}{\iff} \mathrm{Id}_{\mathcal{T}}(e_{\mathcal{T},j}) = e_{\mathcal{T},k}$$

We know that $\mathrm{Id}_{\mathcal{T}}(e_{\mathcal{T},j}) = e_{\mathcal{T},j}$. Therefore for $j \in \mathbb{N}$, $g(j) = j$, and hence g is also the identity function. Therefore $F(\mathrm{Id}_{\mathcal{T}}) = g = \mathrm{Id}_{F(\mathcal{T})} = \mathrm{Id}_{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}$. □

Lemma 5.5. *F preserves composition of morphisms.*

Proof. Let $\mathcal{T}, \mathcal{S}, \mathcal{U}, f_1, f_2 \in \mathbf{FFT}$ such that $f_1 : \mathcal{T} \to \mathcal{S}$ and $f_2 : \mathcal{S} \to \mathcal{U}$. Additionally, let $\{e_{\mathcal{T},0}, e_{\mathcal{T},1}, e_{\mathcal{T},2}, \ldots\}$, $\{e_{\mathcal{S},0}, e_{\mathcal{S},1}, e_{\mathcal{S},2}, \ldots\}$, and $\{e_{\mathcal{U},0}, e_{\mathcal{U},1}, e_{\mathcal{U},2}, \ldots\}$ represent our $\mathcal{T}$-computable, $\mathcal{S}$-computable, and $\mathcal{U}$-computable enumerations of end nodes of $\mathcal{T}$, $\mathcal{S}$, and $\mathcal{U}$ respectively as in Lemma 4.8. Furthermore, assume that for some $j, k, \ell \in \mathbb{N}$:

$$f_1(e_{\mathcal{T},j}) = e_{\mathcal{S},k}$$

$$f_2(e_{\mathcal{S},k}) = e_{\mathcal{U},\ell}$$

Therefore $(f_2 \circ f_1)(e_{\mathcal{T},j}) = e_{\mathcal{U},\ell}$. Now, we know that $F(\mathcal{T}) = \mathcal{A}_{\mathcal{T}_{\mathbb{N}}}$, $F(\mathcal{S}) = \mathcal{A}_{\mathcal{S}_{\mathbb{N}}}$, and $F(\mathcal{U}) = \mathcal{A}_{\mathcal{U}_{\mathbb{N}}}$, so we have that,

$$F(f_1) = g_1 : \mathcal{A}_{\mathcal{T}_{\mathbb{N}}} \to \mathcal{A}_{\mathcal{S}_{\mathbb{N}}}$$

$$F(f_2) = g_2 : \mathcal{A}_{\mathcal{S}_{\mathbb{N}}} \to \mathcal{A}_{\mathcal{U}_{\mathbb{N}}}$$

Now, to check that composition is preserved, we examine $F(f_1)$ and $F(f_2)$ individually first, and then their composition. We have the following.

$$F(f_1)(j) = g_1(j) = k \quad \text{and} \quad F(f_2)(k) = g_2(k) = \ell$$

$$\implies F(f_2 \circ f_1)(j) = (g_2 \circ g_1)(j) = \ell$$

Therefore, $F(f_2 \circ f_1) = (g_2 \circ g_1) = F(f_2) \circ F(f_1)$, which completes our proof. □

Therefore, we have that as defined, F does indeed yield a functor from **FFT** to **NEquiv**.

Theorem 5.6. *Let $F : \boldsymbol{FFT} \to \boldsymbol{NEquiv}$ be defined such that $F(\mathcal{T}) = \mathcal{A}_{\mathcal{T}_{\mathbb{N}}}$ and $F(f) = g$ where $g(j) = k \overset{defn}{\iff} f(e_{\mathcal{T},j}) = e_{\mathcal{S},k}$. Then F is a functor.*

Proof. The proof is contained in the prior lemmas: Lemma 5.3, Lemma 5.4, and Lemma 5.5. □

Now that we have built the functor F, we wish to examine some of its various properties. We show now that this functor behaves quite nicely. It is full, faithful, and "essentially onto". First, we

recall some terminology from category theory. Let $\mathcal{T}, \mathcal{S} \in \mathbf{FFT}$. Then $F_{\mathcal{T},\mathcal{S}}$ denotes the following map.

$$F_{\mathcal{T},\mathcal{S}} : \{f \mid f \in \mathbf{FFT} \wedge f : \mathcal{T} \to \mathcal{S}\} \longrightarrow \{g \mid g \in \mathbf{NEquiv} \wedge g : F(\mathcal{T}) \to F(\mathcal{S})\}$$

We begin first by showing that F is full, that is that the map $F_{\mathcal{T},\mathcal{S}}$ is onto. Note that we are now using the term "full" in two ways from this point on. We recall that a *full tree* is one in which all of its paths are exactly the same length. A *full functor* is one in which the map $F_{\mathcal{T},\mathcal{S}}$ is onto. Though these are very different definitions for the same word, these definitions actually refer to different entities entirely — either trees or functors. The meaning of "full" should therefore be clear from the context.

We introduce here notation for the ith-level predecessor of a node on some tree. Let $p(\mathcal{T}, x, i) =$ the ith level predecessor of node x on tree $\mathcal{T}$. Note that for $\mathcal{T} \in \mathbf{FFT}$, $p(\cdot, \cdot, \cdot)$ is a $\mathcal{T}$-computable function, as in Corollary 4.6 and Corollary 4.4.

Theorem 5.7. *The functor $F : \boldsymbol{FFT} \to \boldsymbol{NEquiv}$ is full.*

Proof. Let $\mathcal{T}, \mathcal{S} \in \mathbf{FFT}$, both of height $n+1$ for some $n \geq 1$, and let g be some isomorphism from $F(\mathcal{T}) = \mathcal{A}_{\mathcal{T}_\mathbb{N}}$ to $F(\mathcal{S}) = \mathcal{A}_{\mathcal{S}_\mathbb{N}}$. To show that $F_{\mathcal{T},\mathcal{S}}$ is onto, we need to show that there exists some morphism $f \in \mathbf{FFT}$ such that $f : \mathcal{T} \to \mathcal{S}$ and $F(f) = g$. Since morphisms in **FFT** are simply ismorphisms between trees, we need to define an isomorphism, $f : \mathcal{T} \to \mathcal{S}$.

Instead of doing just this, let's first define an isomorphism, $\tilde{f} : \mathcal{T}_\mathbb{N} \to \mathcal{S}_\mathbb{N}$. Recall that in Theorem 4.10 and especially Lemma 4.11 we defined $h^{\mathcal{T}}_\mathbb{N} : \mathcal{T} \to \mathcal{T}_\mathbb{N}$ to be an isomorphism. Therefore, $(h^{\mathcal{S}}_\mathbb{N})^{-1} : \mathcal{S}_\mathbb{N} \to \mathcal{S}$ is also an isomorphism. As we can see from Figure 5.2, then, once we define $\tilde{f}$, we can then define f as follows:

$$f \stackrel{\text{defn}}{=} (h^{\mathcal{S}}_\mathbb{N})^{-1} \circ \tilde{f} \circ h^{\mathcal{T}}_\mathbb{N}$$

To show f is indeed an isomorphism, then, all we will need to show is that $\tilde{f}$ is an isomorphism. Then we would have that f is 1-1 and onto since $h^{\mathcal{S}}_\mathbb{N}$, $\tilde{f}$, and $h^{\mathcal{T}}_\mathbb{N}$ are. We would also have that f

preserves order since $h^{\mathcal{S}}_{\mathbb{N}}$, $\tilde{f}$, and $h^{\mathcal{T}}_{\mathbb{N}}$ do:

$$
\begin{aligned}
x \prec_{\mathcal{T}} y &\iff h^{\mathcal{T}}_{\mathbb{N}}(x) \prec_{\mathcal{T}_{\mathbb{N}}} h^{\mathcal{T}}_{\mathbb{N}}(y) && \text{(b/c } h^{\mathcal{T}}_{\mathbb{N}} \text{ an isomorphism)} \\
&\iff \tilde{f}\big(h^{\mathcal{T}}_{\mathbb{N}}(x)\big) \prec_{\mathcal{S}_{\mathbb{N}}} \tilde{f}\big(h^{\mathcal{T}}_{\mathbb{N}}(y)\big) && \text{(b/c } \tilde{f} \text{ an isomorphism)} \\
&\iff (h^{\mathcal{S}}_{\mathbb{N}})^{-1}\Big(\tilde{f}\big(h^{\mathcal{T}}_{\mathbb{N}}(x)\big)\Big) \prec_{\mathcal{S}} (h^{\mathcal{S}}_{\mathbb{N}})^{-1}\Big(\tilde{f}\big(h^{\mathcal{T}}_{\mathbb{N}}(y)\big)\Big) && \text{(b/c } (h^{\mathcal{S}}_{\mathbb{N}})^{-1} \text{ an isomorphism)} \\
&\iff \big((h^{\mathcal{S}}_{\mathbb{N}})^{-1} \circ \tilde{f} \circ h^{\mathcal{T}}_{\mathbb{N}}\big)(x) \prec_{\mathcal{S}} \big((h^{\mathcal{S}}_{\mathbb{N}})^{-1} \circ \tilde{f} \circ h^{\mathcal{T}}_{\mathbb{N}}\big)(y) \\
&\iff f(x) \prec_{\mathcal{S}} f(y)
\end{aligned}
$$

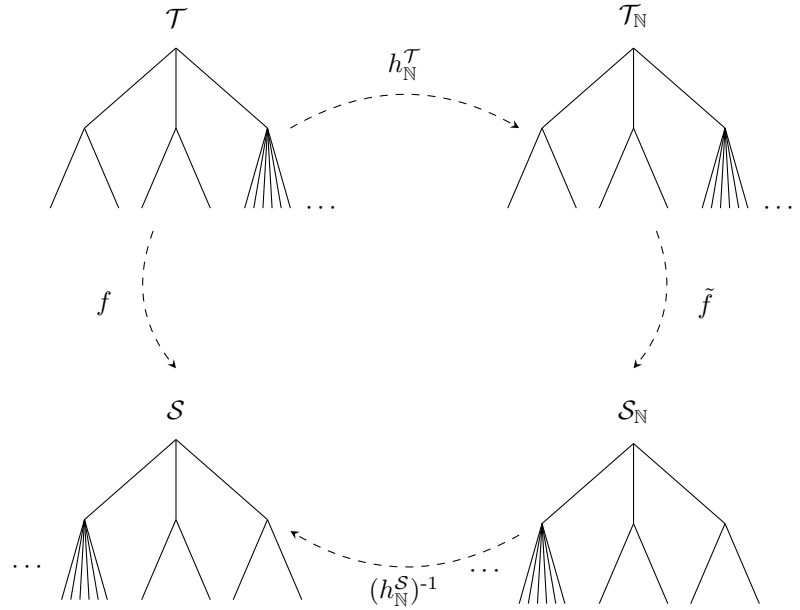


Figure 5.2: Construction of f from $\tilde{f}$ and $h_{\mathbb{N}}$ in proof of Theorem 5.7, that the functor $F : \mathbf{FFT} \to \mathbf{NEquiv}$ is full.

So, we proceed here and define $\tilde{f} : \mathcal{T}_{\mathbb{N}} \to \mathcal{S}_{\mathbb{N}}$ and show it is an isomorphism. First, we recall that nodes of $\mathcal{T}_{\mathbb{N}}$ are of the form $[j]_{E_i^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}}$ for some $j \in \mathbb{N}$ and some $i \in \{0, \ldots, n+1\}$. Also, since $g : \mathbb{N} \to \mathbb{N}$ is an ismorphism, we know $g(j) \in \mathbb{N}$ for any $j \in \mathbb{N}$ (and certainly for those j's which define a node of $\mathcal{T}_{\mathbb{N}}$). Therefore, recalling that each element of the domain of $\mathcal{S}_{\mathbb{N}}$ is represented on the $(n+1)$-th level of $\mathcal{S}_{\mathbb{N}}$ (see Corollary 4.16), we know that $[g(j)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}} \in \mathcal{S}_{\mathbb{N}}$. Therefore, for each $[j]_{E_i^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}} \in T_{\mathbb{N}}$ we can now define $\tilde{f}$:

$$
\tilde{f}([j]_{E_i^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}}) \stackrel{\text{defn}}{=} p(\mathcal{S}_{\mathbb{N}}, [g(j)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}}, i) \tag{5.2}
$$

We recall that the node $p(\mathcal{S}_{\mathbb{N}}, [g(j)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}}, i)$ is defined to be the ith level predecessor on $\mathcal{S}_{\mathbb{N}}$ of $[g(j)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}}$. Note that we do allow $i = n+1$, and therefore we are considering predecessors under $\preceq_{\mathcal{S}_{\mathbb{N}}}$.

We now show that $\tilde{f}$ is 1-1. Let $[j]_{E_i^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}}, [k]_{E_\ell^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}} \in T_{\mathbb{N}}$ such that $\tilde{f}([j]_{E_i^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}}) = \tilde{f}([k]_{E_\ell^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}})$. Then,

$$\tilde{f}([j]_{E_i^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}}) = \tilde{f}([k]_{E_\ell^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}}) \implies p(\mathcal{S}_{\mathbb{N}}, [g(j)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}}, i) = p(\mathcal{S}_{\mathbb{N}}, [g(k)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}}, \ell)$$

Note that nodes of $\mathcal{S}_{\mathbb{N}}$ are labelled uniquely. (No two nodes have the same labelling.) So the only way this can happen is if $i = \ell$.

$$\begin{aligned}
&\implies p(\mathcal{S}_{\mathbb{N}}, [g(j)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}}, i) = p(\mathcal{S}_{\mathbb{N}}, [g(k)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}}, \ell) \\
&\qquad = [a]_{E_i^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}} \quad \text{for some } a \in \mathbb{N} \text{ and } i = \ell
\end{aligned}$$

Since $a \in \mathbb{N}$ and g an isomorphism on $\mathbb{N}$, we also know that $a = g(b)$ for some $b \in \mathbb{N}$.

$$\begin{aligned}
&\implies a = g(b) E_i^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}} g(j) E_i^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}} g(k) E_i^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}} \\
&\qquad \big(\text{by Equation 4.6, since } \exists \text{ node } x = [a]_{E_i^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}} \text{ s.t.} \\
&\qquad x \preceq_{\mathcal{S}_{\mathbb{N}}} [g(j)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}}, [g(k)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}}\big) \\
&\implies b E_i^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}} j E_i^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}} k \quad \text{(since } g \text{ an isomorphism)} \\
&\implies [b]_{E_i^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}} = [j]_{E_i^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}} = [k]_{E_\ell^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}} \text{ as equiv. classes}
\end{aligned}$$

Note that these three equivalence classes are all equal in $\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}$, but this does not mean that they are necessarily all nodes of $\mathcal{T}_{\mathbb{N}}$ (though they are all represented as part of some node in $\mathcal{T}_{\mathbb{N}}$). But in fact, we assumed that both $[j]_{E_i^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}}, [k]_{E_\ell^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}}$ are indeed nodes of $\mathcal{T}_{\mathbb{N}}$. The only way this can happen is if they are identical.

$$\implies j = k \text{ and } i = \ell$$

Therefore $[j]_{E_i^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}} = [k]_{E_\ell^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}}$, and we have completed the proof that $\tilde{f}$ is 1-1.

Now we prove that $\tilde{f}$ is onto. We let $[a]_{E_i^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}}$ be a node in $\mathcal{S}_{\mathbb{N}}$. This means $a \in \mathbb{N}$, and because g is an isomorphism, we know there is a unique $b \in \mathbb{N}$ such that $g(b) = a$. Also, by Corollary 4.16 we

know that $[a]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}} \in S_\mathbb{N}$, $[g(b)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}} \in S_\mathbb{N}$ and $\text{level}_{\mathcal{S}_\mathbb{N}}([a]_{E_i^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}) = i$. Therefore,

$$[a]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}} = p(\mathcal{S}_\mathbb{N}, [g(b)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}, i)$$

Now consider $[b]_{E_i^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}}$ as an equivalence class. Note, we have no guarantee that $[b]_{E_i^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}}$ is a node in $\mathcal{T}_\mathbb{N}$, only that it is represented by some node in $\mathcal{T}_\mathbb{N}$. So, let c be the smallest element in this equivalence class. Then $bE_i^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}c$, and $[c]_{E_i^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}} \in T_\mathbb{N}$. Now, let $g(c) = d$ for some $d \in \mathbb{N}$. Then, because g is an isomorphism and preserves equivalence classes,

$$bE_i^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}c \implies g(b)E_i^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}g(c) \implies aE_i^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}d$$

Therefore, $d \in [a]_{E_i^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}$, and hence $[a]_{E_i^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}$ is the ith level predecessor in $\mathcal{S}_\mathbb{N}$ of $[d]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}$. That is,

$$[a]_{E_i^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}} = p(\mathcal{S}_\mathbb{N}, [g(c)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}, i) = \tilde{f}([c]_{E_i^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}})$$

So, given some node $[a]_{E_i^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}$ in $\mathcal{S}_\mathbb{N}$, we have found a node in $\mathcal{T}_\mathbb{N}$ such that applying $\tilde{f}$ to it gives us back $[a]_{E_i^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}$. Namely, this is node $[c]_{E_i^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}}$ in $\mathcal{T}_\mathbb{N}$, where c is the smallest natural number such that $cE_i^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}g^{-1}(a)$. Therefore $\tilde{f}$ is onto.

Now, we show that $\tilde{f}$ preserves order. Let $[a]_{E_i^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}}, [b]_{E_j^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}} \in T_\mathbb{N}$. Then,

$$\begin{aligned}
[a]_{E_i^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}} \prec_{\mathcal{T}_\mathbb{N}} [b]_{E_j^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}} &\implies bE_i^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}a \text{ and } i < j \\
&\implies g(b)E_i^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}g(a) \\
&\implies \exists\, x \text{ at level } i \text{ s.t. } x \prec_{\mathcal{T}_\mathbb{N}} [g(b)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}} \text{ and } x \prec_{\mathcal{T}_\mathbb{N}} [g(a)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}
\end{aligned}$$

Specifically, this tells us that x is the ith level predecessor of $[g(b)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}$ and $[g(a)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}$ in $\mathcal{A}_{\mathcal{S}_\mathbb{N}}$. In other words,

$$x = p(\mathcal{S}_\mathbb{N}, [g(b)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}, i) = p(\mathcal{S}_\mathbb{N}, [g(a)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}, i) = \tilde{f}([g(a)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}})$$

Now, let y be the jth level predecessor of $[g(b)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}$. Since $i < j$ and both x and y are predecessors of $[g(b)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}$, this implies that $x \prec_{\mathcal{S}_\mathbb{N}} y$. This gives us:

$$\tilde{f}([g(a)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}) = x \prec_{\mathcal{S}_\mathbb{N}} y = p(\mathcal{S}_\mathbb{N}, [g(b)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}, j) = \tilde{f}([g(b)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}})$$

Therefore $[a]_{E_i^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}} \prec_{\mathcal{T}_\mathbb{N}} [b]_{E_j^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}} \implies \tilde{f}([g(a)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}) \prec_{\mathcal{S}_\mathbb{N}} \tilde{f}([g(b)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}})$. Now, to show the other

direction we have:

$$\begin{aligned}\tilde{f}([g(a)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}) \prec_{\mathcal{S}_\mathbb{N}} \tilde{f}([g(b)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}) &\implies i\text{th level predecessor of } [g(a)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}} \\ &\quad \text{“} \prec_{\mathcal{S}_\mathbb{N}} \text{”} \\ &\quad j\text{th level predecessor of } [g(b)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}} \\ &\implies p(\mathcal{S}_\mathbb{N}, [g(a)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}, i) \prec_{\mathcal{S}_\mathbb{N}} p(\mathcal{S}_\mathbb{N}, [g(b)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}, j)\end{aligned}$$

Since clearly $[g(a)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}$ is a successor of its ith level predecessor, and same for $[g(b)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}$ and its jth level predecessor. Therefore we have:

$$p(\mathcal{S}_\mathbb{N}, [g(a)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}, i) \prec_{\mathcal{S}_\mathbb{N}} [g(a)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}, \text{ and,}$$

$$p(\mathcal{S}_\mathbb{N}, [g(a)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}, i) \prec_{\mathcal{S}_\mathbb{N}} p(\mathcal{S}_\mathbb{N}, [g(b)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}, j) \prec_{\mathcal{S}_\mathbb{N}} [g(a)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}$$

This gives that there is a node z at level i, namely node $z = p(\mathcal{S}_\mathbb{N}, [g(a)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}, i)$, such that

$$z \prec_{\mathcal{S}_\mathbb{N}} [g(b)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}} \text{ and } z \prec_{\mathcal{S}_\mathbb{N}} [g(a)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}$$

Therefore, applying Equation 4.6 yields that $g(b) E_i^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}} g(a)$. Since g is an isomorphism, and hence preserves equivalence relations, we know then that $b E_i^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}} a$. Now, considering the below as equivalence classes (not necessarily nodes on $\mathcal{T}_\mathbb{N}$), and recalling that equivalence classes are nested in $\mathcal{A}_{\mathcal{T}_\mathbb{N}}$, we have:

$$\begin{aligned}b E_i^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}} a &\implies b \in [a]_{E_i^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}} = [b]_{E_i^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}} \supseteq [b]_{E_j^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}} \\ &\implies [a]_{E_i^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}} \supseteq [b]_{E_j^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}}\end{aligned}$$

We already know that $[a]_{E_i^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}}$ are indeed nodes $[b]_{E_j^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}}$ in $\mathcal{T}_\mathbb{N}$ (we assumed as much at the beginning of this part of the proof). We also already showed that $i < j$. Therefore applying Equation 4.7 we get that:

$$[a]_{E_i^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}} \prec_{\mathcal{T}_\mathbb{N}} [b]_{E_j^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}}$$

Hence we have now shown that $\tilde{f}([a]_{E_i^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}}) \prec_{\mathcal{S}_\mathbb{N}} \tilde{f}([b]_{E_j^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}}) \implies [a]_{E_i^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}} \prec_{\mathcal{T}_\mathbb{N}} [b]_{E_j^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}}$. Therefore, we have now completed both directions to show that $\tilde{f}$ preserves order.

Since we now showed that $\tilde{f}$ is 1-1, onto, and preservers order, we have completed the proof that $\tilde{f}$

is an isomorphism. Hence building f as previously described will yield the appropriate isomorphism $f : \mathcal{T} \to \mathcal{S}$.

There is only one final part of the proof, and that is to show that as built, $F(f) = g$. So, let $g(j) = k$. We need to show that $F(f)(j) = k$. By the definition in Equation 5.1, $F(f)(j) = k \overset{\text{defn}}{\iff} f(e_{\mathcal{T},j}) = e_{\mathcal{S},k}$. So, we now apply f to $e_{\mathcal{T},j}$:

$$\begin{aligned}
f(e_{\mathcal{T},j}) &= ((h_{\mathbb{N}}^{\mathcal{S}})^{\text{-}1} \circ \tilde{f} \circ h_{\mathbb{N}}^{\mathcal{T}})(e_{\mathcal{T},j}) \\
&= ((h_{\mathbb{N}}^{\mathcal{S}})^{\text{-}1} \circ \tilde{f})\big(h_{\mathbb{N}}^{\mathcal{T}}(e_{\mathcal{T},j})\big) \\
&= ((h_{\mathbb{N}}^{\mathcal{S}})^{\text{-}1} \circ \tilde{f})([j]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}}) \\
&= (h_{\mathbb{N}}^{\mathcal{S}})^{\text{-}1}\big(\tilde{f}([j]_{E_{n+1}^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}})\big) \\
&= (h_{\mathbb{N}}^{\mathcal{S}})^{\text{-}1}\big(p(\mathcal{S}_{\mathbb{N}}, [g(j)]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}}, n+1)\big) \\
&= (h_{\mathbb{N}}^{\mathcal{S}})^{\text{-}1}\big(p(\mathcal{S}_{\mathbb{N}}, [k]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}}, n+1)\big) \quad \text{(recall } g(j) = k \text{ by assumption)} \\
&= (h_{\mathbb{N}}^{\mathcal{S}})^{\text{-}1}([k]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}}) \\
&= e_{\mathcal{S}_{\mathbb{N}},k}
\end{aligned}$$

Therefore $F(f)(j) = k$, and hence $F(f) = g$ as intended. This completes the proof.

□

Now that we have proven that the functor F is full, we have a method to go from any isomorphism between nested equivalence structures $F(\mathcal{T})$ and $F(\mathcal{S})$, to an isomorphism between $\mathcal{T}$ and $\mathcal{S}$. For instance, we now have the following very useful corollary.

Corollary 5.8. *Let $\mathcal{T}, \mathcal{S}$ be trees in **FFT**, with functor $F : \boldsymbol{FFT} \to \boldsymbol{NEquiv}$. Then,*

$$\mathcal{T} \simeq \mathcal{S} \iff F(\mathcal{T}) \simeq F(\mathcal{S})$$

Proof. Let $\mathcal{T}, \mathcal{S} \in \mathbf{FFT}$. First, if $\mathcal{T} \simeq \mathcal{S}$, this means there exists some isomorphism $f \in \mathbf{FFT}$ such that $f : \mathcal{T} \to \mathcal{S}$. Since F is a functor, and specifically as shown in Lemma 5.3, we know then that $F(f) = g$ is an isomorphism from $F(\mathcal{T})$ to $F(\mathcal{S})$. Therefore $F(\mathcal{T}) \simeq F(\mathcal{S})$. Conversely, if $F(\mathcal{T}) \simeq F(\mathcal{S})$, then there must be some isomorphism, $g : F(\mathcal{T}) \to F(\mathcal{S})$. Because, as shown in Theorem 5.7, F is full, or in other words $F_{\mathcal{T},\mathcal{S}}$ is onto, we know then that there is some isomorphism $f : \mathcal{T} \to \mathcal{S}$ such that $F(f) = g$. The existence of this isomorphism guarantees then that $\mathcal{T} \simeq \mathcal{S}$. □

We now continue on and show that our functor $F : \mathbf{FFT} \to \mathbf{NEquiv}$ is faithful, that is that the map $F_{\mathcal{T},\mathcal{S}}$ is 1-1.

Theorem 5.9. *The functor $F : \boldsymbol{FFT} \to \boldsymbol{NEquiv}$ is faithful.*

Proof. Let $\mathcal{T}, \mathcal{S} \in \mathbf{FFT}$, both of height $n+1$ for some $n \geq 1$. Furthermore, let $f_1, f_2 \in \mathbf{FFT}$ such that $F(f_1) = F(f_2) = g$ for some isomorphism $g \in \mathbf{NEquiv}$, and let t be some node at level i of $\mathcal{T}$ for some $i \in \{0, \ldots, n+1\}$. To show $f_1 = f_2$ we will show that $f_1(t) = f_2(t)$ for any input $t \in T$. Since both f_1 and f_2 are isomorphisms onto $\mathcal{S}$ and t is at level i of $\mathcal{T}$, then $f_1(t)$ and $f_2(t)$ are both nodes at level i of $\mathcal{S}$. Let $e_{\mathcal{S},k_1}$ be the least end node in $\mathcal{S}$ that is a successor of $f_1(t)$, and let $e_{\mathcal{S},k_2}$ be the least end node in $\mathcal{S}$ that is a successor of $f_2(t)$. Since we may have that $i = n+1$, the terminating level of the tree, we consider successors under $\preceq_{\mathcal{S}}$.

Since f_1 and f_2 are isomorphisms, this means there exist end nodes in $\mathcal{T}$, which are themselves successors of t, such that when we apply f_1 and f_2 we get $e_{\mathcal{S},k_1}$ and $e_{\mathcal{S},k_2}$ respectively. That is, there exist $j_1, j_2 \in \mathbb{N}$ such that:

$$f_1(e_{\mathcal{T},j_1}) = e_{\mathcal{S},k_1} \quad \text{and} \quad f_1(e_{\mathcal{T},j_2}) = e_{\mathcal{S},k_2}$$

Note that $k_1 < k_2$ does not necessarily imply that $j_1 < j_2$, or vice versa, but we do not need this fact in our proof. Applying our definition of F on morphisms from Equation 5.1,

$$F(f_1) = k_1 \quad \text{and} \quad F(f_2) = k_2$$

We recall that by assumption, $F(f_1) = F(f_2) = g$. Therefore we are talking about the same function, g, in both cases. Hence $g(j_1) = k_1$ and $g(j_2) = k_2$. By the way we've defined $\mathcal{A}_{\mathcal{T}_\mathbb{N}}$ and $\mathcal{A}_{\mathcal{S}_\mathbb{N}}$ as in Theorem 4.24, this means that:

1. Nodes $e_{\mathcal{T},j_1}, e_{\mathcal{T},j_2}$ must be successors of t in $\mathcal{T}$, because their images under isomorphisms f_1 and f_2, respectively, are successors of $f(t)$ in $\mathcal{S}$.

2. Since t is at level i, when we apply $h^{\mathcal{T}}_{\mathbb{N}}$ to generate $\mathcal{T}_\mathbb{N}$, there will be some node in $\mathcal{T}_\mathbb{N}$ corresponding to t, call it $h^{\mathcal{T}}_{\mathbb{N}}(t)$, which has $[j_1]_{E^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}_{n+1}}$ and $[j_1]_{E^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}_{n+1}}$ as successors.

3. Once we apply $\tilde{h}$, then, to get our nested equivalence structure $\mathcal{A}_{\mathcal{T}_\mathbb{N}}$, this yields that j_1, j_2 are in the same $E^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}_i$-th equivalence class. (Applying Equation 4.6 which defines the equivalence relation $E^{\mathcal{A}_{\mathcal{T}_\mathbb{N}}}_i$.)

4. Now, when we apply g, an isomorphism, this means that k_1, k_2 are the images of j_1, j_2 respectively and therefore must be in the same $E_i^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}$ equivalence class.

So, taking $k_1 E_i^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}} k_2$ and again applying Equation 4.6 which now we can view as defining the equivalence relation $E_i^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}$, this means there exists a node x at level i of $\mathcal{S}_\mathbb{N}$ such that $x \prec_{\mathcal{S}_\mathbb{N}} [k_1]_{E_i^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}$ and $x \prec_{\mathcal{S}_\mathbb{N}} [k_2]_{E_i^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}$. Now we note that $h^{\mathcal{S}}_{\mathbb{N}}(f_1(t))$ is:

- at level i of $\mathcal{S}_\mathbb{N}$, because t is at level i of $\mathcal{T}$, and $f_1, h^{\mathcal{S}}_{\mathbb{N}}$ are both ismorphisms; and,
- a predecessor of $[k_1]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}$ in $\mathcal{S}_\mathbb{N}$, by how we defined $\prec_{\mathcal{S}_\mathbb{N}}$ with regards to $h^{\mathcal{S}}_{\mathbb{N}}$ in Equation 4.4, since $e_{\mathcal{S},k_1}$ is an end node in $\mathcal{S}$ which is a successor of $f_1(t)$. (That is, we know $h^{\mathcal{S}}_{\mathbb{N}}\big(f_1(t)\big) \preceq_{\mathcal{S}_\mathbb{N}} [k_1]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}$ since $f_1(t) \preceq_{\mathcal{S}_\mathbb{N}} e_{\mathcal{S},k_1}$.)

Therefore, $x = h^{\mathcal{S}}_{\mathbb{N}}(f_1(t))$, since a node can have only one predecessor at each level. Similarly, we have that $h^{\mathcal{S}}_{\mathbb{N}}(f_2(t))$ is:

- at level i of $\mathcal{S}_\mathbb{N}$, because t is at level i of $\mathcal{T}$, and $f_2, h^{\mathcal{S}}_{\mathbb{N}}$ are both ismorphisms; and,
- a predecessor of $[k_2]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}$ in $\mathcal{S}_\mathbb{N}$, by how we defined $\prec_{\mathcal{S}_\mathbb{N}}$ with regards to $h^{\mathcal{S}}_{\mathbb{N}}$ in Equation 4.4. (That is, we know $h^{\mathcal{S}}_{\mathbb{N}}\big(f_2(t)\big) \preceq_{\mathcal{S}_\mathbb{N}} [k_2]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}}$ since $f_2(t) \preceq_{\mathcal{S}_\mathbb{N}} e_{\mathcal{S},k_2}$.)

Therefore, $x = h^{\mathcal{S}}_{\mathbb{N}}(f_2(t))$, since again, a node can have only one predecessor at each level. Together, this means that $h^{\mathcal{S}}_{\mathbb{N}}(f_1(t)) = h^{\mathcal{S}}_{\mathbb{N}}(f_2(t))$. Recall, we already showed in Lemma 4.12 that $h^{\mathcal{S}}_{\mathbb{N}}$ is 1-1. Therefore $f_1(t) = f_2(t)$. Since we chose t to be some arbitrary node in $\mathcal{T}$, this means that $f_1(t) = f_2(t)$ for any $t \in T$. Therefore $f_1 = f_2$, completing the proof. Hence $F_{\mathcal{T},\mathcal{S}}$ is indeed 1-1. □

Finally, we can show that the functor $F : \mathbf{FFT} \to \mathbf{NEquiv}$ is "essentially onto". Recalling this definition, we show that for every finitely-nested equivalence structure $\mathcal{A} \in \mathbf{NEquiv}$, there is a corresponding full, finite height tree $\mathcal{T} \in \mathbf{FFT}$ such that when we apply the functor F to it, we get a structure $F(\mathcal{T})$ isomorphic to our original structure $\mathcal{A}$. It is this final key property of the functor that, together paired with the fact that we can do this all computably (we will show the algorithmic part of this in Section 5.3), allows us to transfer results from trees of finite height to finitely nested equivalence structures.

At a high-level, the fact that F is essentially onto is easy to see intuitively. We take a nested equivalence structure, $\mathcal{A}$, and draw a tree from it; we could even follow the intuitive method as described in Section 4.3. Then we relabel the nodes in some algorithmic manner so that they're in $\mathbb{N}$; for instance, we could wind our way through the branches to ensure that all the infinitely many nodes get labelled. This new object that we have created will clearly be a tree. Furthermore, we

have in some sense coded the nesting of $\mathcal{A}$ into the branching of this tree. Therefore, when we apply F back to it, we maintain the branching and hence maintain the nesting, i.e. we get the same type of nesting back that we started with. Therefore, this new nested equivalence structure is isomorphic to the original one, even if the labels of the elements are not identical. We now proceed with the formal proof of this fact.

Theorem 5.10. *The functor $F : \boldsymbol{FFT} \to \boldsymbol{NEquiv}$ is essentially onto.*

Proof. Let $\mathcal{A} = (A, E_1, \ldots, E_n)$ be an n-nested equivalence structure in **NEquiv**. Let $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ be the tree built from $\mathcal{A}$ as in Theorem 4.41. We already proved in Corollary 4.37 that $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ is a full finite height tree, with height ≥ 2 for $n \geq 1$ and infinitely many nodes. Therefore $\mathcal{T}_{\mathcal{A}_\mathbb{N}} \in \mathbf{FFT}$. To show that F is essentially onto, we will show then that $F(\mathcal{T}_{\mathcal{A}_\mathbb{N}})$ is isomorphic to $\mathcal{A}$. To do this, we will explicitly build an isomorphism: $g : \mathcal{A} \to F(\mathcal{T}_{\mathcal{A}_\mathbb{N}})$. Figure 5.3 gives an overview of just how we will go about this using the tree $\mathcal{T}_\mathcal{A}$, as built in Theorem 4.27, and the tree $(\mathcal{T}_{\mathcal{A}_\mathbb{N}})_\mathbb{N}$, as built in Theorem 4.10. Recall that as we've defined the functor F, $F(\mathcal{T}_{\mathcal{A}_\mathbb{N}}) = \mathcal{A}_{(\mathcal{T}_{\mathcal{A}_\mathbb{N}})_\mathbb{N}}$.

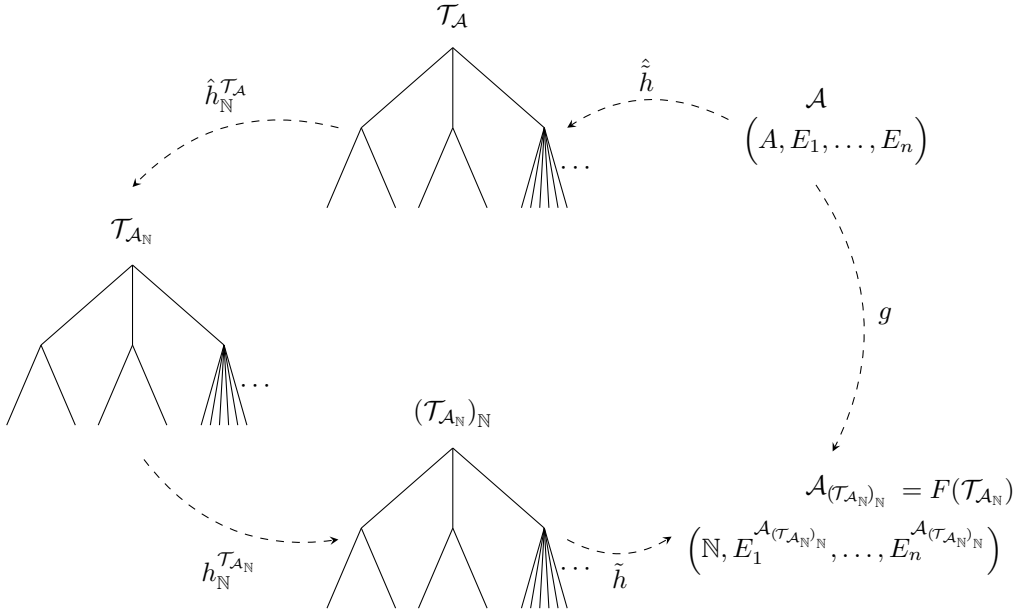


Figure 5.3: Diagram showing steps involved in proof of Theorem 5.10, that the functor $F : \mathbf{FFT} \to \mathbf{NEquiv}$ is essentially onto.

So, let $a \in A$. Define the isomorphism g as follows.

$$g(a) = k \overset{\text{defn}}{\iff} (h_\mathbb{N}^{\mathcal{T}_{\mathcal{A}_\mathbb{N}}} \circ \hat{h}_\mathbb{N}^{\mathcal{T}_\mathcal{A}})([a]_{E_{n+1}^\mathcal{A}}) = [k]_{E_{n+1}^{\mathcal{A}_{(\mathcal{T}_{\mathcal{A}_\mathbb{N}})_\mathbb{N}}}} \tag{5.3}$$

Now, to see that g is 1-1 we let $a, b \in \mathcal{A}$ such that $g(a) = g(b) = k$ for some $k \in \mathbb{N}$. This implies the

following,

$$(h_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}} \circ \hat{h}_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}}})([a]_{E_{n+1}^{\mathcal{A}}}) = (h_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}} \circ \hat{h}_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}}})([b]_{E_{n+1}^{\mathcal{A}}}) = [k]_{E_{n+1}^{\mathcal{A}(\mathcal{T}_{\mathcal{A}_{\mathbb{N}}})_{\mathbb{N}}}}$$

We know from Lemma 4.13 and Lemma 4.36 that $h_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}}$ and $\hat{h}_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}}}$ are both isomorphisms. Therefore their composition is 1-1, and hence $[a]_{E_{n+1}^{\mathcal{A}}} = [b]_{E_{n+1}^{\mathcal{A}}}$, when considered as nodes on the tree $\mathcal{T}_{\mathcal{A}}$. These two nodes can be equal, however, only when $a = b$. Therefore $g(a) = g(b) \implies a = b$, and hence g is 1-1.

Now we prove that g is onto. Let $k \in \mathbb{N}$. We need to show there is some element $a \in A$ such that $g(a) = k$. Based on the definition of g given in Equation 5.3, we examine the node $[k]_{E_{n+1}^{\mathcal{A}(\mathcal{T}_{\mathcal{A}_{\mathbb{N}}})_{\mathbb{N}}}}$ on tree $(\mathcal{T}_{\mathcal{A}_{\mathbb{N}}})_{\mathbb{N}}$. We know that this is a node on the tree by Corollary 4.16. Furthermore, since $h_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}}$ and $\hat{h}_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}}}$ are isomorphisms, they are onto, and hence their composition is onto. Therefore there exists some node on tree $\mathcal{T}_{\mathcal{A}}$ such that:

$$(h_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}} \circ \hat{h}_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}}})([a]_{E_{n+1}^{\mathcal{A}}}) = [k]_{E_{n+1}^{\mathcal{A}(\mathcal{T}_{\mathcal{A}_{\mathbb{N}}})_{\mathbb{N}}}}$$

Furthermore, we know that if $[a]_{E_{n+1}^{\mathcal{A}}}$ is a node on $\mathcal{T}_A$, this means that a must be an element of the universe, A, of our nested equivalence structure, $\mathcal{A}$. Therefore given some $k \in \mathbb{N}$, we know there exists $a \in A$ such that $g(a) = k$, and g is onto.

Finally, we prove that g preserves equivalence relations. Let $a, b \in A$ such that $g(a) = k$ and $g(b) = j$, and let $i \in \{0, \ldots, n+1\}$. Then,

$$\begin{aligned}
aE_i^{\mathcal{A}}b &\iff \exists \text{ node } x \text{ at level } i \text{ of } \mathcal{T}_{\mathcal{A}} \text{ s.t. } x \prec_{\mathcal{T}_{\mathcal{A}}} [a]_{E_{n+1}^{\mathcal{A}}} \text{ and } x \prec_{\mathcal{T}_{\mathcal{A}}} [b]_{E_{n+1}^{\mathcal{A}}} \\
&\qquad \text{(by definition in Equation 4.6)} \\
&\iff \exists \text{ node } y = (h_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}} \circ \hat{h}_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}}})(x) \text{ at level } i \text{ of } \mathcal{T}_{\mathcal{A}} \text{ s.t.} \\
&\qquad\quad y \prec_{(\mathcal{T}_{\mathcal{A}_{\mathbb{N}}})_{\mathbb{N}}} (h_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}} \circ \hat{h}_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}}})([a]_{E_{n+1}^{\mathcal{A}}}) \text{ and } y \prec_{(\mathcal{T}_{\mathcal{A}_{\mathbb{N}}})_{\mathbb{N}}} (h_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}} \circ \hat{h}_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}}})([b]_{E_{n+1}^{\mathcal{A}}}) \\
&\qquad \text{(because } h_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}}, \hat{h}_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}}} \text{ are isomorphisms)} \\
&\iff \exists \text{ node } y \text{ at level } i \text{ of } \mathcal{T}_{\mathcal{A}_{\mathbb{N}}\mathbb{N}} \text{ s.t.} \\
&\qquad\quad y \prec_{(\mathcal{T}_{\mathcal{A}_{\mathbb{N}}})_{\mathbb{N}}} [k]_{E_{n+1}^{\mathcal{A}(\mathcal{T}_{\mathcal{A}_{\mathbb{N}}})_{\mathbb{N}}}} \text{ and } y \prec_{(\mathcal{T}_{\mathcal{A}_{\mathbb{N}}})_{\mathbb{N}}} [j]_{E_{n+1}^{\mathcal{A}(\mathcal{T}_{\mathcal{A}_{\mathbb{N}}})_{\mathbb{N}}}} \\
&\qquad \big(\text{because } g(a) = k \text{ and } g(b) = j\big) \\
&\iff g(a) = kE_i^{\mathcal{A}(\mathcal{T}_{\mathcal{A}_{\mathbb{N}}})_{\mathbb{N}}} j = g(b) \\
&\qquad \text{(by definition in Equation 4.6)}
\end{aligned}$$

Therefore, g preserves equivalence relations, completing the proof that g is indeed an isomorphism. This gives that for any $\mathcal{A} \in \mathbf{NEquiv}$, there exists a tree in **FFT** (namely the tree $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$) such that $F(\mathcal{T}_{\mathcal{A}_\mathbb{N}}) \simeq \mathcal{A}$. Therefore the functor F is essentially onto.

□

Having shown that our functor $F : \mathbf{FFT} \to \mathbf{NEquiv}$ is essentially onto, we can now examine under what conditions two nested equivalence structures are isomorphic. The following very nice corollary is immediate. In some sense it is parallel to Corollary 5.8.

Corollary 5.11. *Let $\mathcal{A}, \mathcal{B}$ be nested equivalence structures in **NEquiv**, and let $\mathcal{T}_{\mathcal{A}_\mathbb{N}}, \mathcal{T}_{\mathcal{B}_\mathbb{N}}$ be as built in Theorem 4.41. Then,*

$$\mathcal{A} \simeq \mathcal{B} \iff \mathcal{T}_{\mathcal{A}_\mathbb{N}} \simeq \mathcal{T}_{\mathcal{B}_\mathbb{N}}$$

Proof. Let $\mathcal{A}, \mathcal{B} \in \mathbf{NEquiv}$, and let $\mathcal{T}_{\mathcal{A}_\mathbb{N}}, \mathcal{T}_{\mathcal{B}_\mathbb{N}}$ be as built in Theorem 4.41. As proved in Theorem 5.10, $F(\mathcal{T}_{\mathcal{A}_\mathbb{N}}) \simeq \mathcal{A}$ and similarly $F(\mathcal{T}_{\mathcal{B}_\mathbb{N}}) \simeq \mathcal{B}$. By Corollary 5.8 we also have that $\mathcal{T}_{\mathcal{A}_\mathbb{N}} \simeq \mathcal{T}_{\mathcal{B}_\mathbb{N}} \iff F(\mathcal{T}_{\mathcal{A}_\mathbb{N}}) \simeq F(\mathcal{T}_{\mathcal{B}_\mathbb{N}})$. Therefore,

$$\mathcal{A} \simeq \mathcal{B} \implies F(\mathcal{T}_{\mathcal{A}_\mathbb{N}}) \simeq \mathcal{A} \simeq \mathcal{B} \simeq F(\mathcal{T}_{\mathcal{B}_\mathbb{N}}) \implies \mathcal{T}_{\mathcal{A}_\mathbb{N}} \simeq \mathcal{T}_{\mathcal{B}_\mathbb{N}}$$

$$\mathcal{T}_{\mathcal{A}_\mathbb{N}} \simeq \mathcal{T}_{\mathcal{B}_\mathbb{N}} \implies \mathcal{A} \simeq F(\mathcal{T}_{\mathcal{A}_\mathbb{N}}) \simeq F(\mathcal{T}_{\mathcal{B}_\mathbb{N}}) \simeq \mathcal{B} \implies \mathcal{A} \simeq \mathcal{B}$$

□

We have now shown that our functor $F : \mathbf{FFT} \to \mathbf{NEquiv}$ exhibits several nice properties. It is full, faithful, and essentially onto. Turning again to the language of category theory, we can now state that this functor constitutes an equivalence of the categories **FFT** and **NEquiv**. We now have enough information to transfer over category-theoretic properties from **FFT** to **NEquiv** and vice versa. While this in and of itself is quite interesting, we are focused here on computability-theoretic notions of **FFT** and **NEquiv**. Exploring whether the computability-theoretic properties transfer from **FFT** to **NEquiv** is the subject of the following section.

5.3 Computability of the Functor

In order for us to examine whether the computability-theoretic properties transfer from **FFT** to **NEquiv** and vice versa, we must explore the computability of the functor, $F : \mathbf{FFT} \to \mathbf{NEquiv}$, and the computability of its various properties. We recall that in all of the work we did in Chapter 4,

we proved that the various maps used along the way were indeed computable, or rather that they were computable relative to our given tree, set, and/or nested equivalence structure. These facts will be very useful here.

Formally, F is said to be a **computable functor** if for $\mathcal{T}, \mathcal{S}, f \in \mathbf{FFT}$ and $f : \mathcal{T} \to \mathcal{S}$, we have that $F(\mathcal{T})$ is uniformly computable in $\mathcal{T}$ and $F(f)$ is uniformly computable in $\mathcal{T} \oplus \mathcal{S} \oplus f$. We now show that this is indeed the case, that the functor F is computable.

Theorem 5.12. *F is a computable functor.*

Proof. Let $\mathcal{T} \in \mathbf{FFT}$. We recall that we defined $F(\mathcal{T}) = \mathcal{A}_{\mathcal{T}_{\mathbb{N}}}$, where $\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}$ is the nested equivalence structure built from tree, $\mathcal{T}$, and set, $\mathbb{N}$, as in Theorem 4.24. We already showed in Lemma 4.21 that $\mathcal{A}_{\mathcal{T}_A}$ is $(\mathcal{T} \oplus A)$-computable for any set $A \subseteq \mathbb{N}$. Furtheremore, the construction in Lemma 4.21 was uniform in $\mathcal{T}$ and A; that is, the same procedure works for any $\mathcal{T}$ and A. Therefore, since $\mathbb{N}$ is a computable set, $\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}$ is uniformly $\mathcal{T}$-computable, and hence $F(\mathcal{T})$ is uniformly $\mathcal{T}$-computable, as desired.

Now, let $\mathcal{T}, \mathcal{S}, f \in \mathbf{FFT}$ and $f : \mathcal{T} \to \mathcal{S}$. Recall we defined $F(f) = g$, where g was defined as in Equation 5.1 and depended upon enumerations of the end nodes of $\mathcal{T}$ and $\mathcal{S}$. So, to show that $F(f)$ is $(\mathcal{T} \oplus \mathcal{S} \oplus f)$-computable, we need to describe a process which, given $j \in \mathbb{N}$ will compute $g(j)$, using only oracles for $\mathcal{T}$, $\mathcal{S}$, and f.

1. Using a $\mathcal{T}$-oracle (as in Lemma 4.8), enumerate end nodes of $\mathcal{T}$ until the jth end node, $e_{\mathcal{T},j}$, has been enumerated.

2. Using an f-oracle, now compute $f(e_{\mathcal{T},j})$.

3. Similarly, using Lemma 4.8 and an $\mathcal{S}$-oracle, enumerate end nodes of $\mathcal{S}$ until we have found $f(e_{\mathcal{T},j})$. That is, enumerate each end node in order and ask: $e_{\mathcal{S},0} = f(e_{\mathcal{T},j})$? $e_{\mathcal{S},1} = f(e_{\mathcal{T},j})$? $e_{\mathcal{S},2} = f(e_{\mathcal{T},j})$? ... until we have found k such that $e_{\mathcal{S},k} = f(e_{\mathcal{T},j})$. Note that since $e_{\mathcal{T},j}$ is an end node of $\mathcal{T}$ and f is an isomorphism, we are guaranteed that $f(e_{\mathcal{T},j})$ is an end node of $\mathcal{S}$, therefore this process will eventually halt.

4. Finally, once we have found this k, output $F(f)(j) = g(j) = k$.

Therefore $F(f)$ is a $(\mathcal{T} \oplus \mathcal{S} \oplus f)$-computable isomorphism from $F(\mathcal{T})$ to $F(S)$, or in other words $F(f) \leq_T \mathcal{T} \oplus \mathcal{S} \oplus f$. Furthermore, the process described was uniform in $\mathcal{T} \oplus \mathcal{S} \oplus f$; that is, the same procedure works for any $\mathcal{T}$, $\mathcal{S}$, and f. Therefore $F(f)$ is uniformly $(\mathcal{T} \oplus \mathcal{S} \oplus f)$-computable, and therefore the functor F is computable. □

We now further examine the computability properties of the isomorphism $F(f)$. We show that having $F(f)$ is enough to compute the isomorphism f.

Corollary 5.13. *Given our functor $F : \boldsymbol{FFT} \to \boldsymbol{NEquiv}$ and isomorphism $f : \mathcal{T} \to \mathcal{S}$, we also have the following:*

$$f \leq_T \mathcal{T} \oplus \mathcal{S} \oplus F(f)$$

Proof. Let $g = F(f)$. To prove this we need to show that given some $t \in T$ and given oracles for $\mathcal{T}$, $\mathcal{S}$, and g we can determine $f(t)$. Below is such a process.

1. First, using a $\mathcal{T}$-oracle, determine $\text{level}_{\mathcal{T}}(t)$, as in Corollary 4.4. Say $\text{level}_{\mathcal{T}}(T) = i$.

2. Then, find the smallest end node successor of t. That is, using a $\mathcal{T}$-oracle enumerate end nodes, $e_{\mathcal{T},j}$, of $\mathcal{T}$ one at a time as in Lemma 4.8. After we have enumerated each one, ask (using our $\mathcal{T}$-oracle): $t \preceq_{\mathcal{T}} e_{\mathcal{T},0}$? $t \preceq_{\mathcal{T}} e_{\mathcal{T},1}$? $t \preceq_{\mathcal{T}} e_{\mathcal{T},2}$? ... Continue on until we have found the smallest such end node, say $t \preceq_{\mathcal{T}} e_{\mathcal{T},\ell}$. (Since t is on our full tree, we are guaranteed to have at least one such end node successor of t.)

3. Now, apply $F(f)$ as appropriate to this end node. Since $F(f)$ is actually a function on $\mathbb{N}$, what we mean by this is: using a g oracle, calculate $g(\ell)$. Say then that $g(\ell) = k$.

4. Using an $\mathcal{S}$-oracle, start enumerating end nodes of $\mathcal{S}$ as in Lemma 4.8 until we have found the kth one, $e_{\mathcal{S},k}$.

5. Find $p(\mathcal{S}, e_{\mathcal{S},k}, i) =$ the ith level predecessor of $e_{\mathcal{S},k}$. Note that this is possible using an $\mathcal{S}$-oracle by Corollary 4.6 and Corollary 4.4. Say that $p(\mathcal{S}, e_{\mathcal{S},k}, i) = s$

6. Finally, halt and output $f(t) = s$.

Therefore, we have used only $\mathcal{T}$, $\mathcal{S}$, and g oracles to calculate f on a given input t. Hence, f is $(\mathcal{T} \oplus \mathcal{S} \oplus F(f))$-computable. □

Clearly, for $\mathcal{T}$ and $\mathcal{S}$ both computable and for $F(f) = g$, then we have that g is f-computable, and f is g-computable. We can extend these properties in a few more useful ways with the following two corollaries.

Corollary 5.14. *Let $\mathcal{T}, \mathcal{S}$ be full finite height trees and let $\mathcal{T}$ be computable. Then,*

$$\mathcal{T} \simeq_{\deg(\mathcal{S})} \mathcal{S} \iff F(T) \simeq_{\deg(\mathcal{S})} F(\mathcal{S})$$

Proof. Let $\mathcal{T}, \mathcal{S} \in \mathbf{FFT}$ such that $\mathcal{T}$ is computable. We first assume that $\mathcal{T} \simeq_{\deg(\mathcal{S})} \mathcal{S}$. This means there is a $\deg(\mathcal{S})$-computable isomorphism $f : \mathcal{T} \to \mathcal{S}$. Furthermore, because F is a functor (as shown in Theorem 5.6) we know that $F(f)$ is itself an isomorphism from $F(\mathcal{T})$ to $F(\mathcal{S})$. Therefore, applying the results of Theorem 5.12, we have:

$$\deg\big(F(f)\big) \leq \deg(\mathcal{T}) \cup \deg(\mathcal{S}) \cup \deg(f) \leq \mathbf{0} \cup \deg(\mathcal{S}) \cup \deg(\mathcal{S})$$

Therefore, $\deg\big(F(f)\big) \leq \deg(\mathcal{S})$, and hence $F(f)$ is a $\deg(\mathcal{S})$-computable isomorphism from $F(\mathcal{T})$ to $F(\mathcal{S})$. This gives that $F(T) \simeq_{\deg(\mathcal{S})} F(\mathcal{S})$.

Next we assume that $F(T) \simeq_{\deg(\mathcal{S})} F(\mathcal{S})$. Therefore there exists some $\deg(\mathcal{S})$-computable isomorphism $g : F(\mathcal{T}) \to F(\mathcal{S})$. Since F is full by Theorem 5.7, this means there exists some corresponding isomorphism $f : \mathcal{T} \to \mathcal{S}$ such that $F(f) = g$. Therefore by Corollary 5.13 we have that:

$$\deg(f) \leq \deg(\mathcal{T}) \cup \deg(\mathcal{S}) \cup \deg\big(F(f)\big) \leq \mathbf{0} \cup \deg(\mathcal{S}) \cup \deg(\mathcal{S})$$

Therefore $\deg(f) \leq \deg(\mathcal{S})$ and $f : \mathcal{T} \to \mathcal{S}$ is an isomorphism computable in $\deg(\mathcal{S})$. This gives that $\mathcal{T} \simeq_{\deg(\mathcal{S})} \mathcal{S}$. □

Corollary 5.15. *Let $\mathcal{T}, \mathcal{S}$ both be computable full finite height trees and let $\mathbf{d}$ be some Turing degree. Then,*

$$\mathcal{T} \simeq_{\mathbf{d}} \mathcal{S} \iff F(T) \simeq_{\mathbf{d}} F(\mathcal{S})$$

Proof. Let $\mathcal{T}, \mathcal{S} \in \mathbf{FFT}$ such that $\mathcal{T}$ and $\mathcal{S}$ are computable. We first assume that $\mathcal{T} \simeq_{\mathbf{d}} \mathcal{S}$. This means there exists an isomorphism $f : \mathcal{T} \to \mathcal{S}$ such that $\deg(f) \leq \mathbf{d}$. Furthermore, since F is a functor, $F(f)$ is also an isomorphism from $F(\mathcal{T})$ to $F(\mathcal{S})$. Therefore by Theorem 5.6 and the fact that $\mathcal{T}$ and $\mathcal{S}$ are both computable:

$$\deg\big(F(f)\big) \leq \deg(\mathcal{T}) \cup \deg(\mathcal{S}) \cup \deg(f) \leq \mathbf{0} \cup \mathbf{0} \cup \mathbf{d}$$

Therefore, $\deg\big(F(f)\big) \leq \mathbf{d}$ and $F(f)$ must be an isomorphism from $F(\mathcal{T})$ to $F(\mathcal{S})$ which is computable in $\mathbf{d}$. Therefore $F(T) \simeq_{\mathbf{d}} F(\mathcal{S})$.

Next, we assume that $F(T) \simeq_{\mathbf{d}} F(\mathcal{S})$. This means there exists some isomorphism $g : F(\mathcal{T}) \to F(\mathcal{S})$ such that g is computable in $\mathbf{d}$. Since F is a full functor, we know there exists some isomorphism $f : \mathcal{T} \to \mathcal{S}$ such that $F(f) = g$. Therefore by Corollary 5.13 and the fact that $\mathcal{T}$ and $\mathcal{S}$ are both

computable we have:

$$\deg(f) \leq \deg(\mathcal{T}) \cup \deg(\mathcal{S}) \cup \deg\big(F(f)\big) \leq \mathbf{0} \cup \mathbf{0} \cup \mathbf{d}$$

Therefore, $\deg(f) \leq \mathbf{d}$, meaning f is a $\mathbf{d}$-computable isomorphism from $\mathcal{T}$ to $\mathcal{S}$, and hence we have that $\mathcal{T} \simeq_{\mathbf{d}} \mathcal{S}$. □

We now extend the property of F being essentially onto, in order to show that F is "$\mathcal{A}$-essentially onto". In other words, we show that for any finitely nested equivalence structure, $\mathcal{A} \in \mathbf{NEquiv}$, there is an associated tree in **FFT** such that when we apply F, we get a nested equivalence structure which is $\mathcal{A}$-computably isomorphic to $\mathcal{A}$.

Corollary 5.16. *Let $\mathcal{A}$ be any n-nested equivalence structure, let $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ be a full finite height tree as built in Theorem 4.41, and let $F : \boldsymbol{FFT} \to \boldsymbol{NEquiv}$ be our functor. Then*

$$F(\mathcal{T}_{\mathcal{A}_\mathbb{N}}) \simeq_{\deg(\mathcal{A})} \mathcal{A}$$

Proof. Note that from Theorem 5.10 we know that for each $\mathcal{A} \in \mathbf{NEquiv}$ there exists some full finite height tree, namely $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$, such that $F(\mathcal{T}_{\mathcal{A}_\mathbb{N}}) \simeq \mathcal{A}$. Furthermore, in the proof of Theorem 5.10 we built an isomorphism, $g : \mathcal{A} \to F(\mathcal{T}_{\mathcal{A}_\mathbb{N}})$. Therefore, to prove the theorem, all we we need to show is that g is $\mathcal{A}$-computable. Given some $a \in A$, we now describe a process to compute $g(a)$.

1. First, using our $\mathcal{A}$-oracle, start enumerating $C_\mathcal{A}$, the equivalence classes of $\mathcal{A}$ without repetition. Continue listing classes in $C_\mathcal{A}$ (including equivalence relations $E_0^\mathcal{A}$ and $E_{n+1}^\mathcal{A}$) until the class $[a]_{E_{n+1}^\mathcal{A}}$ is listed. Say it is the jth equivalence class listed in the enumeration $C_\mathcal{A}$. That is $[a]_{E_{n+1}^\mathcal{A}} = [c_j]_{E_{i_j}^\mathcal{A}}$.

   Note: this implies that $\hat{h}_\mathbb{N}^{\mathcal{T}_\mathcal{A}}([a]_{E_{n+1}^\mathcal{A}}) = j$.

2. Then, given a $(\mathcal{T}_{\mathcal{A}_\mathbb{N}})$-oracle (which we can compute from an $\mathcal{A}$-oracle, by Lemma 4.38), start enumerating end nodes of $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ until we have found j. That is, use Lemma 4.8 to enumerate end nodes of $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ one at a time and ask $e_0 = j$? $e_1 = j$? $e_2 = j$? ...? Continue on until we have found k such that $e_k = j$. Note, we know that j corresponds to an end node because $j = \hat{h}_\mathbb{N}^{\mathcal{T}_\mathcal{A}}([a]_{E_{n+1}^\mathcal{A}})$, $[a]_{E_{n+1}^\mathcal{A}}$ is an end node of $\mathcal{T}_\mathcal{A}$, and $\hat{h}_\mathbb{N}^{\mathcal{T}_\mathcal{A}}$ is an isomorphism. Therefore, this process will stop and find such a k.

3. Finally, halt and output $g(a) = k$.

The above is an $\mathcal{A}$-computable process to compute the isomorphism $g : \mathcal{A} \to F(\mathcal{T}_{\mathcal{A}_\mathbb{N}})$. Therefore we have that $F(\mathcal{T}_{\mathcal{A}_\mathbb{N}}) \simeq_{\deg(\mathcal{A})} \mathcal{A}$. □

We now finish out the remainder of this section by showing that, in the case when their corresponding domains equal to $\mathbb{N}$, finitely nested equivalence structures and their corresponding trees are indeed Turing equivalent (and vice versa). Let $\mathcal{A} = (\mathbb{N}, E_1, \ldots, E_n)$ be an n-nested equivalence structure, let $\mathcal{T} = (\mathbb{N}, \prec_\mathcal{T})$ be a full tree of finite height $n + 1 \geq 2$, and let $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ and $\mathcal{A}_{\mathcal{T}_\mathbb{N}}$ be the corresponding structures built from them as in Theorem 4.41 and Theorem 4.24 respectively.

Theorem 5.17. $\mathcal{A} \equiv_T \mathcal{T}_{\mathcal{A}_\mathbb{N}}$

Proof. We first recall that by Theorem 4.41, $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ is $\mathcal{A}$-computable, therefore $\mathcal{T}_{\mathcal{A}_\mathbb{N}} \leq_T \mathcal{A}$. Therefore, to complete the proof we need only show that $\mathcal{A} \leq_T \mathcal{T}_{\mathcal{A}_\mathbb{N}}$. To show this, we need to show that given an oracle for $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ and given $k, \ell \in \mathbb{N}$, there is some computable process by which we can determine whether $kE_i\ell$ for each $i \in \{0, \ldots, n + 1\}$.

Recall that $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ is not just any tree, but it is a tree that was built from a nested equivalence structure, and therefore it has several nice properties. For instance, the nodes $0, 1, 2, \ldots, n, n + 1$ in $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ correspond with the nodes $[0]_{E_0}$, $[0]_{E_1}$, $[0]_{E_2}$, … , $[0]_{E_n}$, $[0]_{E_{n+1}}$ respectively on $\mathcal{T}_\mathcal{A}$. Below is such a computable process to determine whether $kE_i\ell$ for each $i \in \{0, \ldots, n + 1\}$.

1. Given $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ we first use its nodes to enumerate nodes of $\mathcal{T}_\mathcal{A}$ until we have enumerated the kth and ℓth end node of $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ and all of their respective predecessors (which will correspond to nodes $[k]_{E_{n+1}}$ and $[\ell]_{E_{n+1}}$ on $\mathcal{T}_\mathcal{A}$ and all of their respective predecessors). The process is as follows:

   (a) Using the oracle for $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$, enumerate the 0th end node of $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$. This will correspond to node $[0]_{E_{n+1}}$ on $\mathcal{T}_\mathcal{A}$. Then, using a $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ oracle computably find all of its predecessors and their levels: p_0, p_1, … , p_n. (This is possible as in Corollary 4.4 and Corollary 4.6.) These will correspond to nodes $[0]_{E_0}$, $[0]_{E_1}$, $[0]_{E_2}$, … , $[0]_{E_n}$ respectively on $\mathcal{T}_\mathcal{A}$.

   (b) Using the oracle for $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$, enumerate the first end node of $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$. This will correspond to node $[1]_{E_{n+1}}$ on $\mathcal{T}_\mathcal{A}$. Then, using a $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ oracle computably find all of its predecessors and their levels: p_0, p_1, … , p_n. (This is possible as in Corollary 4.4 and Corollary 4.6.) If any p_j's are not yet enumerated into $\mathcal{T}_\mathcal{A}$, we enumerate them as node $[1]_{E_j}$ in $\mathcal{T}_\mathcal{A}$.

   (c) Continue on in this manner until we have enumerated $[k]_{E_{n+1}}$ and $[\ell]_{E_{n+1}}$ into $\mathcal{T}_\mathcal{A}$ and their predecessors.

(d) That is, move on to the mth end-node of $\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}$, which will correspond to node $[m]_{E_{n+1}}$ on $\mathcal{T}_{\mathcal{A}}$. Then using a $\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}$ oracle, computably find all of its predecessors and their levels: p_0, $p_1, \ldots, p_n$ (as in Corollary 4.4 and Corollary 4.6). If any p_j's are not yet enumerated into $\mathcal{T}_{\mathcal{A}}$, enumerate each as node $[m]_{E_j}$ in $\mathcal{T}_{\mathcal{A}}$. Continue until we have done this for $m = k$ and $m = \ell$.

2. Now compare the ith level predecessor of $[k]_{E_{n+1}}$ on $\mathcal{T}_{\mathcal{A}}$ with the ith level predecessor of $[\ell]_{E_{n+1}}$ on $\mathcal{T}_{\mathcal{A}}$. If they're the same, then $kE_i\ell$. If they're not the same then $\neg(kE_i\ell)$.

Therefore, the above process uses a $\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}$-oracle to computably determine the equivalence relations in $\mathcal{A} = (\mathbb{N}, E_1, \ldots, E_n)$. Therefore $\mathcal{A} \leq_T \mathcal{T}_{\mathcal{A}_{\mathbb{N}}}$. Hence if $\mathcal{A}$ has domain $\mathbb{N}$, then $\mathcal{A} \equiv_T \mathcal{T}_{\mathcal{A}_{\mathbb{N}}}$. □

Note that in the above theorem, we needed to know ahead of time that the domain of $\mathcal{A}$ was $\mathbb{N}$. The process would have worked equally as well for any infinite domain $A \subseteq \mathbb{N}$, substituting a_k and a_ℓ for k and ℓ in the above proof where $A = \{a_0 < a_1 < \ldots\}$. However, we would have needed to know A ahead of time. For instance, consider nested equivalence structures $\mathcal{A} = (\mathbb{N}, E_1, \ldots, E_n)$ and $\mathcal{B} = (\mathbb{N} - \{0\}, R_1, \ldots, R_n)$ defined by: $aE_ib \iff (a+1)R_i(b+1)$. It is easy to see that if we follow the process given in Theorem 4.41, then $\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}$ and $\mathcal{T}_{\mathcal{B}_{\mathbb{N}}}$ would be identical. Without knowing ahead of time, then, their respective domains, it is impossible to recover $\mathrm{dom}(\mathcal{A})$ and $\mathrm{dom}(\mathcal{B})$ from the associated tree, although we can recover the overall structure of the equivalence relations and their nesting properties.

We now see that a similar result holds for a given tree, $\mathcal{T} = (\mathbb{N}, \prec_{\mathcal{T}})$ which is full and of finite height $n + 1 \geq 2$, and its corresponding nested equivalence structure $\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}$ as built in Theorem 4.24.

Theorem 5.18. $\mathcal{T} \equiv_T \mathcal{A}_{\mathcal{T}_{\mathbb{N}}}$

Proof. We already showed in Lemma 4.21 that $\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}$ is $\mathcal{T}$ computable, and hence $\mathcal{A}_{\mathcal{T}_{\mathbb{N}}} \leq_T \mathcal{T}$. Therefore, to complete the proof we need only show that $\mathcal{T} \leq_T \mathcal{A}_{\mathcal{T}_{\mathbb{N}}}$. To do this, we must show that given an oracle for $\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}$ and given $j, k \in \mathbb{N}$, there is a computable process to determine whether $j \prec_{\mathcal{T}} k$. Below is such a process.

1. Using $\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}$ oracle, start enumerating equivalence classes $C_{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}$ in order until we have enumerated the jth and kth equivalence classes. Call them $[c_j]_{E_{i_j}^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}}$ and $[c_k]_{E_{i_k}^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}}$ respectively for some $c_j, c_k \in \mathbb{N}$ and $i_j, i_k \in \{0, \ldots, n+1\}$.

2. Check if $i_j > i_k$. If yes, then stop; we know that $j \not\prec_{\mathcal{T}} k$. If no, then continue on to the next step.

3. Then, using an $\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}$ oracle, ask whether $c_j E_{i_j}^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}} c_k$.

   - If yes, this implies that as equivalence classes $[c_j]_{E_{i_j}^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}} = [c_k]_{E_{i_j}^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}}$. Hence, $[c_j]_{E_{i_j}^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}} \supseteq [c_k]_{E_{i_k}^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}}$, and $[c_k]_{E_{i_k}^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}} \prec_{\mathcal{T}_{\mathbb{N}}} [c_j]_{E_{i_j}^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}}$. Therefore we know that $j \prec_{\mathcal{T}} k$.
   - If no, this implies that $c_j \notin [c_k]_{E_{i_j}^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}} \supseteq [c_k]_{E_{i_k}^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}}$, hence $[c_k]_{E_{i_k}^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}} \not\prec_{\mathcal{T}} [c_j]_{E_{i_j}^{\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}}}$. Therefore we know that $j \not\prec_{\mathcal{T}} k$.

Therefore, following the above process and using an oracle for $\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}$ we can computably determine $\prec_{\mathcal{T}}$. Therefore $\mathcal{T} \leq_T \mathcal{A}_{\mathcal{T}_{\mathbb{N}}}$. Hence, if $\mathcal{T}$ has domain $\mathbb{N}$, then $\mathcal{T} \equiv_T \mathcal{A}_{\mathcal{T}_{\mathbb{N}}}$. □

Now that we have shown that the functor $F : \mathbf{FFT} \to \mathbf{NEquiv}$ and the associated structures $\mathcal{A}_{\mathcal{T}_{\mathbb{N}}}$ and $\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}$ abide by many nice computability-theoretic properties, in the next chapter we will leverage the existence of this computable functor to obtain many nice computability-theoretic results.

Chapter 6

Computability-Theoretic Properties of Nested Equivalence Structures and Full Trees of Finite Height

Using the framework we have setup in Chapter 4 and Chapter 5, we can now examine many nice computability-'theoretic results pertaining to nested equivalence structures and certain types of full finite height trees. We examine previous work conducted on these types of structures and transfer the results accordingly.

6.1 Trees of Finite Height

Trees of finite height were studied extensively by Lempp, McCoy, Miller, and Solomon in [31]. Although their investigation took into account all trees of finite height, many of the results are clearly directly applicable to full trees of finite height ≥ 2.

Lempp, McCoy, Miller, and Solomon came up with the following structural criterion to help classify the computable categoricity of trees of finite height. Let $\mathcal{T} = (T, \prec_{\mathcal{T}})$ be a tree of finite height and let x be some node in $\mathcal{T}$. Then we let S_x denote the set of immediate successors of x.

That is:

$$S_x = \{y \in T : y \succ_{\mathcal{T}} x \ \wedge \ \neg\exists z(y \succ_{\mathcal{T}} z \succ_{\mathcal{T}} x)\} = \{x_i \in T : i \in I_x\} \text{ for some indexing set } I_x$$

(Note that in general, S_x and I_x are not necessarily computable.) We furthermore define the immediate successor subtree of x_i, to be $\mathcal{T}[x_i] = \{y \in T : x_i \preceq_{\mathcal{T}} y\}$. Then x is defined to be a **node of strongly finite type** by the following conditions:

- If x is a terminating node of tree $\mathcal{T}$, then x is of strongly finite type.
- If x is not a terminating node, and x meets all of the following conditions, then x is of strongly finite type.
    i. There are only finitely many isomorphism types in the set $\{\mathcal{T}[x_i] : i \in I_x\}$, each of which is of strongly finite type.
    ii. For each $j, k \in I$, if $\mathcal{T}[x_j]$ embeds into $\mathcal{T}[x_k]$, then either:
        – $\mathcal{T}[x_j]$ and $\mathcal{T}[x_k]$ are isomorphic; or
        – The isomorphism type of $\mathcal{T}[x_k]$ appears finitely often in the set $\{\mathcal{T}[x_i] : i \in I_x\}$.

A tree, $\mathcal{T}$, is said then to be a **tree of strongly finite type** if every node in $\mathcal{T}$ is of strongly finite type.

Lempp, McCoy, Miller, and Solomon also defined a similar, though slightly weaker condition. The node x is defined to be a **node of finite type** as follows:

- If x is a terminating node of tree $\mathcal{T}$, then x is of finite type.
- If x is not a terminating node, and x meets all of the following conditions, then x is of finite type.
    i. There are only finitely many isomorphism types in the set $\{\mathcal{T}[x_i] : i \in I_x\}$, each of which is of finite type.
    ii. Every isomorphism type which appears infinitely often in the set $\{\mathcal{T}[x_i] : i \in I_x\}$ is of strongly finite type.
    iii. For each $j, k \in I$, if $\mathcal{T}[x_j]$ embeds into $\mathcal{T}[x_k]$, then either:
        – $\mathcal{T}[x_j]$ and $\mathcal{T}[x_k]$ are isomorphic; or
        – The isomorphism type of $\mathcal{T}[x_j]$ appears finitely often in the set $\{\mathcal{T}[x_i] : i \in I_x\}$; or

– The isomorphism type of $\mathcal{T}[x_k]$ appears finitely often in the set $\{\mathcal{T}[x_i] : i \in I_x\}$.

A tree, $\mathcal{T}$, is said then to be a **tree of finite type** if every node in $\mathcal{T}$ is of finite type.

Note that in general it can be difficult to determine whether or not a given node is a terminating node in the first place. Specifically, even for a computable tree, $\mathcal{T}$, $\text{level}_{\mathcal{T}}$ is only a c.e. function. In our context, however, we are dealing with full finite height trees, and therefore determining the level of a node x on tree $\mathcal{T}$ is a $\mathcal{T}$-computable process, as already shown in Lemma 4.3 and Corollary 4.4. This means then that for the most part, determining whether a given node or a given tree is of finite type in the first place will likely be an "easier" process for full trees of finite height than for the general case of trees of finite height.

Lempp, McCoy, Miller, and Solomon used this structural criterion as a way to fully classify computable categoricity and relative computable categoricity of full trees of finite height, with the following theorem.

Theorem 6.1 (Lempp, McCoy, Miller, and Solomon, [31])**.** *Let $\mathcal{T}$ be a computable tree of finite height. Then the following are equivalent:*

1. *$\mathcal{T}$ is a tree of finite type*
2. *$\mathcal{T}$ is computably categorical*
3. *$\mathcal{T}$ is relatively computably categorical*

In addition, these researchers were able to fully classify the possible computable dimensions of full trees of finite height.

Theorem 6.2 (Lempp, McCoy, Miller, and Solomon, [31])**.** *Let $\mathcal{T}$ be a computable tree of finite height. Then the computable dimension of $\mathcal{T}$ is equal to 1 or ω.*

6.2 Computable Categoricity of Nested Equivalence Structures

We can now take the results on computable categoricity of trees of finite height and transfer them to nested equivalence structures. We do this by showing that various properties of computable categoricity hold for a nested equivalence structure $\mathcal{A}$ exactly when they hold for the corresponding tree of finite height.

Lemma 6.3. *Let $\mathcal{A}$ be an n-nested equivalence structure, and let $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ be its corresponding tree built as in Theorem 4.41. Then,*

$$\mathcal{A} \text{ is computably categorical} \iff \mathcal{T}_{\mathcal{A}_\mathbb{N}} \text{ is computably categorical}$$

Proof. We begin first by assuming that $\mathcal{A}$ is computably categorical. Note that if $\mathcal{A}$ is computable, then $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ is also computable by Lemma 4.38. So, to show that $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ is computably categorical, we let $\mathcal{S}$ be some computable tree isomorphic to $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$. We need to show that $\mathcal{S}$ is computably isomorphic to $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$. Note that in the proof of Theorem 5.10 we showed that $F(\mathcal{T}_{\mathcal{A}_\mathbb{N}}) \simeq \mathcal{A}$, and in Corollary 5.8 we showed that $F(\mathcal{T}_{\mathcal{A}_\mathbb{N}}) \simeq F(\mathcal{S})$. Therefore,

$$\mathcal{A} \simeq F(\mathcal{T}_{\mathcal{A}_\mathbb{N}}) \simeq F(\mathcal{S})$$

Since $\mathcal{A}$ is computably categorical by assumption, this gives us the following:

$$F(\mathcal{S}) \simeq_c \mathcal{A} \simeq_c F(\mathcal{T}_{\mathcal{A}_\mathbb{N}})$$

Therefore $F(\mathcal{S})$ and $F(\mathcal{T}_{\mathcal{A}_\mathbb{N}})$ are computably isomorphic. (We can simply compose the computable isomorphism between $F(\mathcal{S})$ and $\mathcal{A}$ with the one between $\mathcal{A}$ and $F(\mathcal{T}_{\mathcal{A}_\mathbb{N}})$.) Therefore there exists some computable isomorphism $g : F(\mathcal{T}_{\mathcal{A}_\mathbb{N}}) \to F(\mathcal{S})$. Since F is full as in Theorem 5.7, there exists some isomorphism $f : \mathcal{T}_{\mathcal{A}_\mathbb{N}} \to \mathcal{S}$ such that $F(f) = g$, and furthermore due to Corollary 5.13 we know that $f \leq_T \mathcal{T}_{\mathcal{A}_\mathbb{N}} \oplus \mathcal{S} \oplus g$. Therefore, since $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$, $\mathcal{S}$, and g are all computable, this means that f is itself a computable isomorphism between $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ and $\mathcal{S}$. Therefore $\mathcal{T}_{\mathcal{A}_\mathbb{N}} \simeq_c \mathcal{S}$. Since $\mathcal{S}$ was chosen arbitrarily, this means that $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ is computably categorical. Therefore if $\mathcal{A}$ is computably categorical, then $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ is computably categorical.

Now, we assume that $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ is computably categorical, and we let $\mathcal{A}$ be computable. To show that $\mathcal{A}$ is computably categorical, we also let $\mathcal{B}$ be some computable nested equivalence structure isomorphic to $\mathcal{A}$. We need to show that $\mathcal{B}$ is computably isomorphic to $\mathcal{A}$. First we note that by Lemma 4.38 $\mathcal{T}_{\mathcal{B}_\mathbb{N}}$ is computable since $\mathcal{B}$ is. We also know by Corollary 5.11 that $\mathcal{T}_{\mathcal{A}_\mathbb{N}} \simeq \mathcal{T}_{\mathcal{B}_\mathbb{N}}$. Additionally, because we are assuming that $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ is computably categorical, we know that we actually have $\mathcal{T}_{\mathcal{A}_\mathbb{N}} \simeq_c \mathcal{T}_{\mathcal{B}_\mathbb{N}}$. We can now apply Corollary 5.15 with $\mathbf{d} = \mathbf{0}$, which yields that $F(\mathcal{T}_{\mathcal{A}_\mathbb{N}}) \simeq_c F(\mathcal{T}_{\mathcal{B}_\mathbb{N}})$. Furthermore, we can also apply Corollary 5.16 which gives that $\mathcal{A} \simeq_c F(\mathcal{T}_{\mathcal{A}_\mathbb{N}})$ and $\mathcal{B} \simeq_c F(\mathcal{T}_{\mathcal{B}_\mathbb{N}})$. Putting this all together yields the following:

$$\mathcal{A} \simeq_c F(\mathcal{T}_{\mathcal{A}_\mathbb{N}}) \simeq_c F(\mathcal{T}_{\mathcal{B}_\mathbb{N}}) \simeq_c \mathcal{B}$$

This means that $\mathcal{A}$ and $\mathcal{B}$ are computably isomorphic as desired. (We can compose the computable isomorphisms between $\mathcal{A}$ and $F(\mathcal{T}_{\mathcal{A}_\mathbb{N}})$, $F(\mathcal{T}_{\mathcal{A}_\mathbb{N}})$ and $F(\mathcal{T}_{\mathcal{B}_\mathbb{N}})$, and $F(\mathcal{T}_{\mathcal{B}_\mathbb{N}})$ and $\mathcal{B}$.) Therefore, since $\mathcal{B}$ was chosen arbitrarily, this means that $\mathcal{A}$ is computably categorical. Hence if $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ is computably categorical, then so, too, is $\mathcal{A}$. □

We now show that relative computable categoricity for a nested equivalence structure coincides with relative computable categoricity of its corresponding full finite height tree.

Lemma 6.4. *Let $\mathcal{A}$ be a computable n-nested equivalence structure, and let $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ be its corresponding tree built as in Theorem 4.41. Then,*

$$\mathcal{A} \text{ is relatively computably categorical} \iff \mathcal{T}_{\mathcal{A}_\mathbb{N}} \text{ is relatively computably categorical}$$

Proof. Note that if $\mathcal{A}$ is computable, then $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ is also computable by Lemma 4.38. We first assume that $\mathcal{A}$ is relatively computably categorical. To show that $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ is relatively computably categorical, we let $\mathcal{S}$ be some (not necessarily computable) tree isomorphic to $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$. We need to show then that $\mathcal{S}$ is $\deg(\mathcal{S})$-isomorphic to $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$. By Corollary 5.8 we know then that $F(\mathcal{T}_{\mathcal{A}_\mathbb{N}}) \simeq F(\mathcal{S})$. Furthermore, by Corollary 5.16 and because we know that $\mathcal{A}$ is computable, we know that $F(\mathcal{T}_{\mathcal{A}_\mathbb{N}}) \simeq_c \mathcal{A}$. Therefore, we have that:

$$\mathcal{A} \simeq_c F(\mathcal{T}_{\mathcal{A}_\mathbb{N}}) \simeq F(\mathcal{S})$$

Since therefore $\mathcal{A}$ and $F(\mathcal{S})$ must be isomorphic, and $\mathcal{A}$ is relatively computably categorical by assumption, this yields that $\mathcal{A} \simeq_{\deg(\mathcal{S})} F(\mathcal{S})$. Putting this all together gives:

$$F(\mathcal{S}) \simeq_{\deg(\mathcal{S})} \mathcal{A} \simeq_c F(\mathcal{T}_{\mathcal{A}_\mathbb{N}})$$

Therefore $F(\mathcal{S}) \simeq_{\deg(\mathcal{S})} F(\mathcal{T}_{\mathcal{A}_\mathbb{N}})$. (We can compose the computable isomorphism between $\mathcal{A}$ and $F(\mathcal{T}_{\mathcal{A}_\mathbb{N}})$ with the $\deg(\mathcal{S})$-computable isomorphism between $F(\mathcal{S})$ and $\mathcal{A}$.) Applying Corollary 5.14 then yields that $\mathcal{T}_{\mathcal{A}_\mathbb{N}} \simeq_{\deg(\mathcal{S})} \mathcal{S}$, as desired. Therefore if $\mathcal{A}$ is relatively computably categorical, so, too, is $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$.

We now assume that $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ is relatively computably categorical. Let $\mathcal{B}$ be some (not necessarily computable) nested equivalence structure isomorphic to $\mathcal{A}$. We need to show then that $\mathcal{B}$ is $\deg(\mathcal{B})$-isomorphic to $\mathcal{A}$. Applying Corollary 5.11 we have that $\mathcal{T}_{\mathcal{A}_\mathbb{N}} \simeq \mathcal{T}_{\mathcal{B}_\mathbb{N}}$. Since $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ is relatively computably categorical by assumption, this yields then that $\mathcal{T}_{\mathcal{A}_\mathbb{N}} \simeq_{\deg(\mathcal{T}_{\mathcal{B}_\mathbb{N}})} \mathcal{T}_{\mathcal{B}_\mathbb{N}}$. Then, applying Corollary 5.14 gives us that $F(\mathcal{T}_{\mathcal{A}_\mathbb{N}}) \simeq_{\deg(\mathcal{T}_{\mathcal{B}_\mathbb{N}})} F(\mathcal{T}_{\mathcal{B}_\mathbb{N}})$. Finally, we can apply Corollary 5.16 to get

the following:

$$\mathcal{A} \simeq_{\deg(\mathcal{A})} F(\mathcal{T}_{\mathcal{A}_\mathbb{N}}) \simeq_{\deg(\mathcal{T}_{\mathcal{B}_\mathbb{N}})} F(\mathcal{T}_{\mathcal{B}_\mathbb{N}}) \simeq_{\deg(\mathcal{B})} \mathcal{B}$$

Note that $\mathcal{A}$ is computable by assumption, and $\mathcal{T}_{\mathcal{B}_\mathbb{N}} \leq_T \mathcal{B}$ by Lemma 4.38. Therefore there is an isomorphism from $\mathcal{A}$ to $\mathcal{B}$ which is computable in $\deg(\mathcal{B})$. (We can simply compose the above isomorphisms together to produce a $\deg(\mathcal{B})$-computable isomorphism.) Therefore $\mathcal{A} \simeq_{\deg(\mathcal{B})} \mathcal{B}$ as desired. Hence if $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ is relatively computably categorical, then $\mathcal{A}$ is also relatively computably categorical. □

We can now take the prior two results on finitely nested equivalence structures and combine it with those obtained by Lempp, McCoy, Miller, and Solomon in [31] on trees of finite height to yield a very nice classification of computable categoricity for finitely nested equivalence structures.

Theorem 6.5. *Let $\mathcal{A}$ be a computable n-nested equivalence structure, and let $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ be its corresponding tree built as in Theorem 4.41. Then the following are equivalent:*

1. *$\mathcal{A}$ is computably categorical*
2. *$\mathcal{A}$ is relatively computably categorical*
3. *$\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ is computably categorical*
4. *$\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ is relatively computably categorical*
5. *$\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ is a tree of finite type*

Proof. We combine the results obtained in Theorem 6.1 with those in Lemma 6.3 and Lemma 6.4. □

We now can also additionally explore the number of computable isomorphism classes of a finitely nested equivalence structure. We examine the computable dimension of such a structure in relation to the computable dimension of its corresponding tree.

Lemma 6.6. *Let $\mathcal{A}$ be a computable n-nested equivalence structure, and let $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ be its corresponding tree built as in Theorem 4.41. Then the computable dimension of $\mathcal{A}$ = the computable dimension of $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$.*

Proof. We first assume that $\mathcal{A}$ has finite computable dimension m. This means that there exist computable nested equivalence structures $\mathcal{B}_1, \ldots, \mathcal{B}_m$ to represent each of the m-many computable isomorphism classes of $\mathcal{A}$. That is, there exist finitely nested equivalence structures $\mathcal{B}_1, \ldots, \mathcal{B}_m$ such that:

i. $\mathcal{B}_1, \ldots, \mathcal{B}_m \simeq \mathcal{A}$

ii. $\mathcal{B}_1, \ldots, \mathcal{B}_m$ are computable

iii. $\mathcal{B}_i \not\simeq_c \mathcal{B}_j$ for $i \neq j$

iv. $\forall$ computable $\mathcal{B}$ such that $\mathcal{B} \simeq \mathcal{A}$, then $\mathcal{B} \simeq_c \mathcal{B}_i$ for some $i \in \{1, \ldots, m\}$

Therefore to show $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ also has computable dimension m we need to find exactly m-many computable isomorphism classes of $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$. Consider the trees $\mathcal{T}_{\mathcal{B}_{1_\mathbb{N}}}$, ... , $\mathcal{T}_{\mathcal{B}_{m_\mathbb{N}}}$. Then we have the following:

i. From Corollary 5.11 we know that $\mathcal{B}_1, \ldots, \mathcal{B}_m \simeq \mathcal{A} \iff \mathcal{T}_{\mathcal{B}_{1_\mathbb{N}}}, \ldots, \mathcal{T}_{\mathcal{B}_{m_\mathbb{N}}} \simeq \mathcal{T}_{\mathcal{A}_\mathbb{N}}$.

ii. For each $i \in \{1, \ldots, m\}$ we know from Lemma 4.38 that $\mathcal{T}_{\mathcal{B}_{i_\mathbb{N}}} \leq_T \mathcal{B}_i$. Therefore, since each $\mathcal{B}_i$ is computable, this means that each $\mathcal{T}_{\mathcal{B}_{i_\mathbb{N}}}$ is also computable.

iii. Let $i \neq j$ and assume to the contrary that $\mathcal{T}_{\mathcal{B}_{i_\mathbb{N}}} \simeq_c \mathcal{T}_{\mathcal{B}_{j_\mathbb{N}}}$. Then by Corollary 5.14 we know that $F(\mathcal{T}_{\mathcal{B}_{i_\mathbb{N}}}) \simeq_c F(\mathcal{T}_{\mathcal{B}_{j_\mathbb{N}}})$. By Corollary 5.16 we also have that:

$$\mathcal{B}_i \simeq_c F(\mathcal{T}_{\mathcal{B}_{i_\mathbb{N}}}) \simeq_c F(\mathcal{T}_{\mathcal{B}_{j_\mathbb{N}}}) \simeq_c \mathcal{B}_j$$

This implies that $\mathcal{B}_i \simeq_c \mathcal{B}_j$, which contradicts our assumption iii. above that $\mathcal{B}_i \not\simeq_c \mathcal{B}_j$ for $i \neq j$. Therefore $\mathcal{T}_{\mathcal{B}_{i_\mathbb{N}}} \not\simeq_c \mathcal{T}_{\mathcal{B}_{j_\mathbb{N}}}$.

iv. Let $\mathcal{S}$ be some computable tree such that $\mathcal{S} \simeq \mathcal{T}_{\mathcal{A}_\mathbb{N}}$. By Corollary 5.8 and Corollary 5.16 we know that:

$$F(\mathcal{S}) \simeq F(\mathcal{T}_{\mathcal{A}_\mathbb{N}}) \simeq \mathcal{A}$$

Therefore, $F(\mathcal{S})$ is some nested equivalence structure which is isomorphic to $\mathcal{A}$. Therefore, by assumption since $\mathcal{A}$ has computable dimension m with representatives $\mathcal{B}_1, \ldots, \mathcal{B}_m$, we know that $F(\mathcal{S}) \simeq_c \mathcal{B}_i$ for some $i \in \{1, \ldots, m\}$. Therefore, also applying Corollary 5.16 again, we have that:

$$F(\mathcal{S}) \simeq_c \mathcal{B}_i \simeq_c F(\mathcal{T}_{\mathcal{B}_{i_\mathbb{N}}})$$

Therefore $F(\mathcal{S}) \simeq_c F(\mathcal{T}_{\mathcal{B}_{i_\mathbb{N}}})$, and applying Corollary 5.14 yields then that $\mathcal{S} \simeq_c \mathcal{T}_{\mathcal{B}_{i_\mathbb{N}}}$.

Hence, by i.-iv. we know that $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ also has exactly m-many computable isomorphism classes, and hence the computable dimension of $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ is also m.

We now assume that the computable dimension of $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ is m and show that the computable dimension of $\mathcal{A}$ is also m. Since the computable dimension of $\mathcal{T}_{\mathcal{A}_\mathbb{N}} = m$, this means that there exist

trees $\mathcal{S}_1, \ldots, \mathcal{S}_m$ to represent each of the m-many computable isomorphism classes of $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$. That is, there exist finite height trees $\mathcal{S}_1, \ldots, \mathcal{S}_m$ such that:

i. $\mathcal{S}_1, \ldots, \mathcal{S}_m \simeq \mathcal{T}_{\mathcal{A}_\mathbb{N}}$

ii. $\mathcal{S}_1, \ldots, \mathcal{S}_m$ are computable

iii. $\mathcal{S}_i \not\simeq_c \mathcal{S}_j$ for $i \neq j$

iv. $\forall$ computable $\mathcal{S}$ such that $\mathcal{S} \simeq \mathcal{T}_{\mathcal{A}_\mathbb{N}}$, then $\mathcal{S} \simeq_c \mathcal{S}_i$ for some $i \in \{1, \ldots, m\}$

Therefore to show $\mathcal{A}$ also has computable dimension m we need to find exactly m-many computable isomorphism classes of $\mathcal{A}$. Consider the nested equivalence structures $F(\mathcal{S}_1)$, ... , $F(\mathcal{S}_m)$. Then we have the following:

i. By Corollary 5.16 we know that $\mathcal{A} \simeq F(\mathcal{T}_{\mathcal{A}_\mathbb{N}})$. Furthermore, by Corollary 5.8 we know that for each $i \in \{1, \ldots, m\}$ $F(\mathcal{S}_i) \simeq F(\mathcal{T}_{\mathcal{A}_\mathbb{N}})$. Therefore, for each i

$$F(\mathcal{S}_i) \simeq F(\mathcal{T}_{\mathcal{A}_\mathbb{N}}) \simeq \mathcal{A}$$

ii. By Theorem 5.12 we know that $F(\mathcal{S}_i)$ is $\mathcal{S}_i$ computable for each $i \in \{1, \ldots, m\}$. Therefore, since $\mathcal{S}_1, \ldots, \mathcal{S}_m$ are all computable by assumption, so too are $F(\mathcal{S}_1), \ldots, F(\mathcal{S}_m)$.

iii. Let $i \neq j$ and assume to the contrary that $F(\mathcal{S}_i) \simeq_c F(\mathcal{S}_j)$ By Corollary 5.14 this means that $\mathcal{S}_i \simeq_c \mathcal{S}_j$, a contradiction to our assumption above. Therefore $F(\mathcal{S}_i) \not\simeq_c F(\mathcal{S}_j)$.

iv. Let $\mathcal{B}$ be some computable nested equivalence structure isomorphic to $\mathcal{A}$. Applying Corollary 5.16 and the fact that $\mathcal{B}$ is computable, we have that

$$F(\mathcal{T}_{\mathcal{B}_\mathbb{N}}) \simeq_c \mathcal{B} \simeq \mathcal{A} \simeq F(\mathcal{T}_{\mathcal{A}_\mathbb{N}})$$

We can now apply Corollary 5.8 and get that $\mathcal{T}_{\mathcal{B}_\mathbb{N}} \simeq \mathcal{T}_{\mathcal{A}_\mathbb{N}}$. Furthermore, since $\mathcal{B}$ is computable, then by Lemma 4.38 $\mathcal{T}_{\mathcal{B}_\mathbb{N}}$ is also computable. Therefore $\mathcal{T}_{\mathcal{B}_\mathbb{N}}$ is a computable tree isomorphic to $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$. Since $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ has computable dimension m with representatives $\mathcal{S}_1, \ldots, \mathcal{S}_m$, this means that $\mathcal{T}_{\mathcal{B}_\mathbb{N}} \simeq_c \mathcal{S}_i$ for some $i \in \{1, \ldots, m\}$. Now, applying Corollary 5.14 yields that $F(\mathcal{T}_{\mathcal{B}_\mathbb{N}}) \simeq_c F(\mathcal{S}_i)$, and therefore

$$\mathcal{B} \simeq_c F(\mathcal{T}_{\mathcal{B}_\mathbb{N}}) \simeq_c F(\mathcal{S}_i)$$

Hence $\mathcal{B}$ is computably isomorphic to $F(\mathcal{S}_i)$ for some $i \in \{1, \ldots, m\}$.

Therefore, by the above we know that $\mathcal{A}$ has exactly m-many computable isomorphism classes, and hence the computable dimension of $\mathcal{A}$ is also m.

We have now shown that for finite computable dimensions, the computable dimension of $\mathcal{A}$ is equal to the computable dimension of $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$. Note, though, that there was nothing special about the given proof which required m to be finite. If instead by assumption $\mathcal{A}$ or $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ had computable dimension equal to ω, we could have easily substituted representatives "$\mathcal{B}_1, \mathcal{B}_2, \ldots$" and "$\mathcal{S}_1, \mathcal{S}_2, \ldots$" for the computable isomorphism classes of $\mathcal{A}$ and $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ respectively, and changed our proof to show "for each $i \in \{1, 2, \ldots\}$" instead of "for each $i \in \{1, \ldots, m\}$ as we showed. The rest of the proof would remain identical.

Therefore, $\mathcal{A}$ and $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ have the same computable dimension, regardless of whether that computable dimension is finite or infinite. □

Now that we know that the computable dimension of a nested equivalence structure is identical to the computable dimension of its corresponding tree, we can now apply results obtained in [31] by Lempp, McCoy, Miller, and Solomon about the computable dimension of finite height trees to inform us about the computable dimension of finitely nested equivalence structures.

Theorem 6.7. *Let $\mathcal{A}$ be a computable finitely nested equivalence structure. Then the computable dimension of $\mathcal{A}$ must be equal to 1 or ω.*

Proof. We apply the results from Theorem 6.2 and Lemma 6.6. □

We can also examine whether finitely nested equivalence structures are **d**-computably categorical, for various Turing degrees, **d**. We first investigate the relationship between the categoricity spectra of finitely nested equivalence structures and their corresponding trees of finite height.

Theorem 6.8. *Let $\mathcal{A}$ be a computable finitely nested equivalence structure, and let $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ be its corresponding tree built as in Theorem 4.41. Then,*

$$\mathit{CatSpec}(\mathcal{A}) = \mathit{CatSpec}(\mathcal{T}_{\mathcal{A}_\mathbb{N}})$$

Proof. We first let $\mathbf{d} \in \mathrm{CatSpec}(\mathcal{A})$. This means that $\mathcal{A}$ is **d**-computably categorical. We wish to show that **d** is also contained in $\mathrm{CatSpec}(\mathcal{T}_{\mathcal{A}_\mathbb{N}})$. Let $\mathcal{S}$ be some computable tree isomorphic to $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$. If $\mathcal{S}$ is computable, then by Lemma 4.21 so, too, is $F(\mathcal{S})$. Applying Corollary 5.13 and Corollary 5.8 then gives

$$\mathcal{A} \simeq_c F(\mathcal{T}_{\mathcal{A}_\mathbb{N}}) \simeq F(\mathcal{S})$$

Since $\mathcal{A}$ is $\mathbf{d}$-computably categorical by assumption, $\mathcal{A}$ and $F(\mathcal{S})$ must be isomorphic via a $\mathbf{d}$-computable isomorphism. Therefore,

$$F(\mathcal{T}_{\mathcal{A}_\mathbb{N}}) \simeq_c \mathcal{A} \simeq_{\mathbf{d}} F(\mathcal{S})$$

Therefore, $F(\mathcal{T}_{\mathcal{A}_\mathbb{N}}) \simeq_{\mathbf{d}} F(\mathcal{S})$, and we can apply Corollary 5.15 to get that $\mathcal{T}_{\mathcal{A}_\mathbb{N}} \simeq_{\mathbf{d}} \mathcal{S}$. Therefore, since $\mathcal{S}$ was chosen arbitrarily, $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ is indeed $\mathbf{d}$-computably categorical, and hence $d \in \mathrm{CatSpec}(\mathcal{T}_{\mathcal{A}_\mathbb{N}})$

Now we let $\mathbf{d} \in \mathrm{CatSpec}(\mathcal{T}_{\mathcal{A}_\mathbb{N}})$ and show this implies that $\mathbf{d} \in \mathrm{CatSpec}(\mathcal{A})$. Let $\mathcal{B}$ be some computable finitely nested equivalence structure isomorphic to $\mathcal{A}$. Therefore by Lemma 4.21 $\mathcal{T}_{\mathcal{B}_\mathbb{N}}$ is computable, and by Corollary 5.11 we know that $\mathcal{T}_{\mathcal{A}_\mathbb{N}} \simeq \mathcal{T}_{\mathcal{B}_\mathbb{N}}$. Since $\mathbf{d} \in \mathrm{CatSpec}(\mathcal{T}_{\mathcal{A}_\mathbb{N}})$ this means that $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ must be $\mathbf{d}$-computably categorical, and hence in fact $\mathcal{T}_{\mathcal{A}_\mathbb{N}} \simeq_{\mathbf{d}} \mathcal{T}_{\mathcal{B}_\mathbb{N}}$. We can now apply Corollary 5.15 and Corollary 5.16 to get that:

$$\mathcal{B} \simeq_c F(\mathcal{T}_{\mathcal{B}_\mathbb{N}}) \simeq_{\mathbf{d}} F(\mathcal{T}_{\mathcal{A}_\mathbb{N}}) \simeq_c \mathcal{A}$$

This yields that $\mathcal{B} \simeq_{\mathbf{d}} \mathcal{A}$. Therefore, since $\mathcal{B}$ was chosen arbitrarily, $\mathcal{A}$ must be $\mathbf{d}$-computably categorical and hence $\mathbf{d} \in \mathrm{CatSpec}(\mathcal{A})$. Therefore $\mathrm{CatSpec}(\mathcal{A}) = \mathrm{CatSpec}(\mathcal{T}_{\mathcal{A}_\mathbb{N}})$. □

6.3 Turing Degree Spectra of Nested Equivalence Structures

We now turn our attention to examining finitely nested equivalence structures and their corresponding full finite height trees with regards to the spectrum of the Turing degrees of each structure.

Theorem 6.9. *Let $\mathcal{A}$ be a finitely nested equivalence structure with domain $\mathbb{N}$, and let $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ be its corresponding tree built as in Theorem 4.41. Then,*

$$DgSp(\mathcal{A}) = DgSp(\mathcal{T}_{\mathcal{A}_\mathbb{N}})$$

Proof. Let $\mathcal{A}$ be some (not necessarily computable) finitely nested equivalence structure with domain $\mathbb{N}$, and let $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ be its corresponding $\mathcal{A}$-computable tree as built in Theorem 4.41. We first let $\mathbf{b} \in \mathrm{DgSp}(\mathcal{A})$. That is, $\mathbf{b} = \deg(\mathcal{B})$ for some nested equivalence structure $\mathcal{B}$ such that $\mathcal{B} \simeq \mathcal{A}$. Note that by Theorem 5.17, $\mathcal{B} \equiv_T \mathcal{T}_{\mathcal{B}_\mathbb{N}}$, and therefore $\mathbf{b} = \deg(\mathcal{B}) = \deg(\mathcal{T}_{\mathcal{B}_\mathbb{N}})$. By Corollary 5.11 we know that $\mathcal{T}_{\mathcal{B}_\mathbb{N}} \simeq \mathcal{T}_{\mathcal{A}_\mathbb{N}}$. Therefore $\deg(\mathcal{T}_{\mathcal{B}_\mathbb{N}}) = \mathbf{b} \in \mathrm{DgSp}(\mathcal{T}_{\mathcal{A}_\mathbb{N}})$.

We now let $\mathbf{s} \in \mathrm{DgSp}(\mathcal{T}_{\mathcal{A}_\mathbb{N}})$. That is $\mathbf{s} = \deg(\mathcal{S})$ for some full finite height tree $\mathcal{S}$ such that $\mathcal{S} \simeq \mathcal{T}_{\mathcal{A}_\mathbb{N}}$. By Theorem 5.18 we know that $\mathcal{S} \equiv_T \mathcal{A}_{\mathcal{S}_\mathbb{N}} = F(\mathcal{S})$, for our functor $F : \mathbf{FFT} \to \mathbf{NEquiv}$.

Therefore $\mathbf{s} = \deg(\mathcal{S}) = \deg(F(\mathcal{S}))$. By Corollary 5.8 and Corollary 5.16 we know that:

$$\mathcal{A}_{\mathcal{S}_{\mathbb{N}}} = F(\mathcal{S}) \simeq F(\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}) \simeq \mathcal{A}$$

Therefore, $F(\mathcal{S}) \simeq \mathcal{A}$ and hence $\mathbf{s} = \deg(F(\mathcal{S})) \in \mathrm{DgSp}(\mathcal{A})$. Therefore, $\mathrm{DgSp}(\mathcal{A}) = \mathrm{DgSp}(\mathcal{T}_{\mathcal{A}_{\mathbb{N}}})$. $\square$

In addition to examining the Turing degree spectrum of finitely nested equivalence structures and their corresponding trees, we can also examine the Turing degree spectrum of relations on finitely nested equivalence structures and their corresponding trees.

We can think of a relation, R, on a finitely nested equivalence structure $\mathcal{A}$ as a set of natural numbers. We say then that the relation R "holds" on a if $a \in R$. We can then define a corresponding relation $\hat{R}$ on $\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}$ as being a relation on end nodes of $\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}$; that is $\hat{R}$ = the set of end nodes of $\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}$ which correspond to the 1-element E_{n+1}-equivalence classes of $\mathcal{A}$ for which R holds. For a more formal definition, we use the tree $\mathcal{T}_{\mathcal{A}}$ and the notation from Section 4.5. Let $\mathcal{A} = (\mathbb{N}, E_1, \ldots, E_n)$ be a finitely nested equivalence structure, and let R be some relation on $\mathcal{A}$. We define relations $\hat{R}^{\mathcal{T}_{\mathcal{A}}}$ and $\hat{R}$ on trees $\mathcal{T}_{\mathcal{A}}$ and $\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}$ respectively as follows:

$$[a]_{E_i} \in \hat{R}^{\mathcal{T}_{\mathcal{A}}} \overset{\text{defn}}{\iff} a \in R \text{ and } i = n+1 \tag{6.1}$$

$$\begin{aligned} x \in \hat{R} &\overset{\text{defn}}{\iff} (\hat{h}_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}}})^{-1}(x) \in \hat{R}^{\mathcal{T}_{\mathcal{A}}} \\ &\iff (\hat{h}_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}}})^{-1}(x) = [a]_{E_{n+1}} \text{ and } a \in R \end{aligned} \tag{6.2}$$

Similarly, if we are given some relation $\hat{R}$ on end nodes of $\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}$, we can define a corresponding relation R on elements of $\mathcal{A}$, such that R holds for a exactly when the corresponding end node of $\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}$ holds under $\hat{R}$. Given a finitely nested equivalence structure $\mathcal{A}$ and a relation $\hat{R}$ on end nodes of $\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}$, we formally define relations $\hat{R}^{\mathcal{T}_{\mathcal{A}}}$ on $\mathcal{T}_{\mathcal{A}}$ and R on $\mathcal{A}$ from $\hat{R}$ as follows. We again use the tree $\mathcal{T}_{\mathcal{A}}$ and the notation as set up in Section 4.5.

$$[a]_{E_{n+1}} \in \hat{R}^{\mathcal{T}_{\mathcal{A}}} \overset{\text{defn}}{\iff} \hat{h}_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}}}([a]_{E_{n+1}}) \in \hat{R} \tag{6.3}$$

$$a \in R \overset{\text{defn}}{\iff} [a]_{E_{n+1}} \in \hat{R}^{\mathcal{T}_{\mathcal{A}}} \tag{6.4}$$

We now note that the relations R, $\hat{R}^{\mathcal{T}_{\mathcal{A}}}$, and $\hat{R}$ are all Turing equivalent.

Lemma 6.10. *Let $\mathcal{A} = (\mathbb{N}, E_1, \ldots, E_n)$ be a computable finitely nested equivalence structure, and let $\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}$ be its corresponding tree built as in Theorem 4.41. Furthermore, let R be some relation on*

$\mathcal{A}$ and let $\hat{R}^{\mathcal{T}_\mathcal{A}}$ and $\hat{R}$ be relations on $\mathcal{T}_\mathcal{A}$ and $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ respectively, as defined in Equations 6.1 and 6.2. Then,

$$R \equiv_T \hat{R}^{\mathcal{T}_\mathcal{A}} \equiv_T \hat{R}$$

Proof. The proof is straightforward from Equations 6.1 to 6.4. We show $\hat{R}^{\mathcal{T}_\mathcal{A}} \equiv_T R$ and $\hat{R} \equiv_T \hat{R}^{\mathcal{T}_\mathcal{A}}$.

We first show $\hat{R}^{\mathcal{T}_\mathcal{A}} \leq_T R$. Given some $[a]_{E_i} \in \mathcal{T}_\mathcal{A}$, to see if $[a]_{E_i} \in \hat{R}^{\mathcal{T}_\mathcal{A}}$, we first ask whether $i = n+1$. If not, then $[a]_{E_i} \notin \hat{R}^{\mathcal{T}_\mathcal{A}}$. If yes, then we ask our R-oracle whether $a \in R$. If not, then $[a]_{E_i} \notin \hat{R}^{\mathcal{T}_\mathcal{A}}$. If yes, then $[a]_{E_i} \in \hat{R}^{\mathcal{T}_\mathcal{A}}$.

It is also clear that $R \leq_T \hat{R}^{\mathcal{T}_\mathcal{A}}$. Given some $a \in \mathbb{N}$, to see whether a $\in R$, we simply ask our $\hat{R}^{\mathcal{T}_\mathcal{A}}$-oracle whether $[a]_{E_{n+1}} \in \hat{R}^{\mathcal{T}_\mathcal{A}}$. If yes, then $a \in R$. If no, then $a \notin R$.

We now show that $\hat{R} \leq_T \hat{R}^{\mathcal{T}_\mathcal{A}}$. Given some $x \in \mathbb{N}$, to see whether $x \in \hat{R}$, we first calculate $\text{level}_{\mathcal{T}_{\mathcal{A}_\mathbb{N}}}(x)$. This is computable by Lemma 4.3. If $\text{level}_{\mathcal{T}_{\mathcal{A}_\mathbb{N}}}(x) \neq n+1$, then $x \notin \hat{R}$. If $\text{level}_{\mathcal{T}_{\mathcal{A}_\mathbb{N}}}(x) = n+1$, then calculate $(\hat{h}_\mathbb{N}^{\mathcal{T}_\mathcal{A}})^{-1}(x)$. This is computable by Lemma 4.33. Now, using our $\hat{R}^{\mathcal{T}_\mathcal{A}}$-oracle, ask $(\hat{h}_\mathbb{N}^{\mathcal{T}_\mathcal{A}})^{-1}(x) \in \hat{R}^{\mathcal{T}_\mathcal{A}}$. If yes, then $x \in \hat{R}$. If no, then $x \notin \hat{R}$.

Finally, it is clear that $\hat{R}^{\mathcal{T}_\mathcal{A}} \leq_T \hat{R}$. Given some $[a]_{E_i} \in \mathcal{T}_\mathcal{A}$, to see if $[a]_{E_i} \in \hat{R}^{\mathcal{T}_\mathcal{A}}$, we first check if $i = n+1$. If not, then $[a]_{E_i} \notin \hat{R}^{\mathcal{T}_\mathcal{A}}$. If yes, then calculate $\hat{h}_\mathbb{N}^{\mathcal{T}_\mathcal{A}}([a]_{E_i})$. This is computable by Lemma 4.33. Now ask our $\hat{R}$-oracle whether $\hat{h}_\mathbb{N}^{\mathcal{T}_\mathcal{A}}([a]_{E_i}) \in \hat{R}$. If yes, then $[a]_{E_i} \in \hat{R}^{\mathcal{T}_\mathcal{A}}$. If not, then $[a]_{E_i} \notin \hat{R}^{\mathcal{T}_\mathcal{A}}$.

□

We now show that the Turing degree spectrum of a relation on a finitely nested equivalence structure is identical to that of its corresponding tree and corresponding relation.

Theorem 6.11. *Let $\mathcal{A} = (\mathbb{N}, E_1, \ldots, E_n)$ be a computable finitely nested equivalence structure, and let $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ be its corresponding tree built as in Theorem 4.41. Furthermore, let R be some relation on $\mathcal{A}$ and let $\hat{R}$ be its corresponding relation on $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ as defined in Equation 6.1. Then,*

$$DgSp_\mathcal{A}(R) = DgSp_{\mathcal{T}_{\mathcal{A}_\mathbb{N}}}(\hat{R})$$

Proof. The proof is contained in the following lemmas: Lemma 6.12 and Lemma 6.13. □

Lemma 6.12. *Let $\mathcal{A} = (\mathbb{N}, E_1, \ldots, E_n)$ be a computable finitely nested equivalence structure, and let $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ be its corresponding tree built as in Theorem 4.41. Furthermore, let R be some relation on $\mathcal{A}$ and let $\hat{R}$ be its corresponding relation on $\mathcal{T}_{\mathcal{A}_\mathbb{N}}$ as defined in Equation 6.1. Then,*

$$DgSp_\mathcal{A}(R) \subseteq DgSp_{\mathcal{T}_{\mathcal{A}_\mathbb{N}}}(\hat{R})$$

Proof. We let $\mathbf{d} \in \mathrm{DgSp}_{\mathcal{A}}(R)$. This means that there exists some computable nested equivalence structure $\mathcal{B} = (\mathbb{N}, E_1^{\mathcal{B}}, \ldots, E_n^{\mathcal{B}})$ which is isomorphic to $\mathcal{A}$ via some isomorphism g under which $\deg\big(g(R)\big) = \mathbf{d}$. Now, to show that $\mathbf{d} \in \mathrm{DgSp}_{\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}}(\hat{R})$, we need to show there exists some computable tree isomorphic to $\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}$ via some isomorphism f such that $\deg\big(f(\hat{R})\big) = \mathbf{d}$. We will show that $\mathcal{T}_{\mathcal{B}_{\mathbb{N}}}$ is such a tree. First note that by construction, $\mathcal{T}_{\mathcal{B}_{\mathbb{N}}}$ has domain $\mathbb{N}$. Furthermore, since $\mathcal{B}$ is computable, so too is $\mathcal{T}_{\mathcal{B}_{\mathbb{N}}}$ (by Lemma 4.38). Additionally, by Corollary 5.11, $\mathcal{T}_{\mathcal{A}_{\mathbb{N}}} \simeq \mathcal{T}_{\mathcal{B}_{\mathbb{N}}}$ since $\mathcal{A} \simeq \mathcal{B}$. Therefore, we need only build an appropriate isomorphism $f : \mathcal{T}_{\mathcal{A}_{\mathbb{N}}} \to \mathcal{T}_{\mathcal{B}_{\mathbb{N}}}$ which yields a degree $\mathbf{d}$ image of the relation. We will build such an f out of g.

Before doing so, we will first define an isomorphism $\tilde{f} : \mathcal{T}_{\mathcal{A}} \to \mathcal{T}_{\mathcal{B}}$. Recall that $p(\mathcal{T}_{\mathcal{B}}, [g(a)]_{E_{n+1}^{\mathcal{B}}}, i) =$ the ith level predecessor on $\mathcal{T}_{\mathcal{B}}$ of $[g(a)]_{E_{n+1}^{\mathcal{B}}}$. We define $\tilde{f}$ as follows.

$$\tilde{f}([a]_{E_i}) \stackrel{\text{defn}}{=} p(\mathcal{T}_{\mathcal{B}}, [g(a)]_{E_{n+1}^{\mathcal{B}}}, i)$$

We now need to show that $\tilde{f}$ is indeed an isomorphism. First, to show $\tilde{f}$ is 1-1, we let $[a]_{E_i}$, $[b]_{E_j}$ be nodes in $\mathcal{T}_{\mathcal{A}}$.

$$\begin{aligned}
\tilde{f}([a]_{E_i}) = \tilde{f}([b]_{E_j}) &\implies p(\mathcal{T}_{\mathcal{B}}, [g(a)]_{E_{n+1}^{\mathcal{B}}}, i) = p(\mathcal{T}_{\mathcal{B}}, [g(b)]_{E_{n+1}^{\mathcal{B}}}, j) \\
&\implies i = j \text{ and } g(a) = g(b) \quad (\text{since } \mathcal{T}_{\mathcal{B}} \text{ nodes labeled uniquely}) \\
&\implies a = b \quad (\text{since } g \text{ an isomorphism and } \therefore \text{1-1}) \\
&\implies [a]_{E_i} = [b]_{E_j}
\end{aligned}$$

Now, to show that $\tilde{f}$ is onto, we let $[b]_{E_j^{\mathcal{B}}}$ be a node in $\mathcal{T}_{\mathcal{B}}$. We need to find a node x in $\mathcal{T}_{\mathcal{A}}$ for which $\tilde{f}(x) = [b]_{E_j^{\mathcal{B}}}$. Since $[b]_{E_j^{\mathcal{B}}}$ is a node in $\mathcal{T}_{\mathcal{B}}$, this means that b is some element in $\mathcal{B}$. Since $g : \mathcal{A} \to \mathcal{B}$ is an isomorphism, this means that there exists a unique element $a \in \mathcal{A}$ such that $g(a) = b$. Now, we let c be the least element in $\mathcal{A}$ such that $cE_j a$. This means that as equivalence classes $[a]_{E_j} = [c]_{E_j}$, and furthermore, $[c]_{E_j} \in C_{\mathcal{A}} = T_{\mathcal{A}}$. We now consider $g(c)$. Since $cE_j a$ and g preserves equivalence relations, we know that $g(c)E_j^{\mathcal{B}} g(a)$. Therefore $g(a) = b \in [g(c)]_{E_j^{\mathcal{B}}}$, when viewed as an equivalence class. This yields:

$$p(\mathcal{T}_{\mathcal{B}}, [g(a)]_{E_{n+1}^{\mathcal{B}}}, j) = p(\mathcal{T}_{\mathcal{B}}, [g(c)]_{E_{n+1}^{\mathcal{B}}}, j) = p(\mathcal{T}_{\mathcal{B}}, [b]_{E_{n+1}^{\mathcal{B}}}, j) = [b]_{E_j^{\mathcal{B}}}$$

Recall that $[b]_{E_j^{\mathcal{B}}} \in \mathcal{T}_{\mathcal{B}}$ by assumption. So, we let $x = [c]_{E_j}$. This gives that $\tilde{f}([c]_{E_j}) = p(\mathcal{T}_{\mathcal{B}}, [b]_{E_{n+1}^{\mathcal{B}}}, j) = [b]_{E_j^{\mathcal{B}}}$, as desired.

Finally, we need to show that $\tilde{f}$ preserves order. To do this we let $[a]_{E_i}$, $[b]_{E_j}$ be nodes in $\mathcal{T}_{\mathcal{A}}$ such that $[a]_{E_i} \prec_{\mathcal{T}_{\mathcal{A}}} [b]_{E_j}$. Since $[a]_{E_i} \prec_{\mathcal{T}_{\mathcal{A}}} [b]_{E_j}$ we know that $i < j$ and that aE_ib. Since g is an isomorphism and therefore preserves equivalence relations, this means that $g(a)E_i^{\mathcal{B}}g(b)$. We let c be the least element which is $E_i^{\mathcal{B}}$-equivalent to $g(a)$. Hence c is also the least element which is $E_i^{\mathcal{B}}$-equivalent to $g(b)$. Therefore,

$$\tilde{f}([a]_{E_i}) = p(\mathcal{T}_{\mathcal{B}}, [g(a)]_{E^{\mathcal{B}}_{n+1}}, i) = [c]_{E_i^{\mathcal{B}}} = p(\mathcal{T}_{\mathcal{B}}, [g(b)]_{E^{\mathcal{B}}_{n+1}}, i)$$

$$\prec_{\mathcal{T}_{\mathcal{B}}} p(\mathcal{T}_{\mathcal{B}}, [g(b)]_{E^{\mathcal{B}}_{n+1}}, j) = \tilde{f}([b]_{E_j})$$

Therefore $[a]_{E_i} \prec_{\mathcal{T}_{\mathcal{A}}} [b]_{E_j} \implies \tilde{f}([a]_{E_i}) \prec_{\mathcal{T}_{\mathcal{B}}} \tilde{f}([b]_{E_j})$, and hence $\tilde{f}$ preserves order. Therefore $\tilde{f}$ is indeed an isomorphism.

Now that we have defined our isomophism $\tilde{f} : \mathcal{T}_{\mathcal{A}} \to \mathcal{T}_{\mathcal{B}}$, we can define our desired isomorphism $f : \mathcal{T}_{\mathcal{A}_{\mathbb{N}}} \to \mathcal{T}_{\mathcal{B}_{\mathbb{N}}}$ which will yield a degree $\mathbf{d}$ image of the relation. We do so by using $\hat{h}$ to go between $\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}, \mathcal{T}_{\mathcal{B}_{\mathbb{N}}}$ and $\mathcal{T}_{\mathcal{A}}, \mathcal{T}_{\mathcal{B}}$ respectively, as can be seen in the diagram in Figure 6.1. Formally, we define $f = \hat{h}_{\mathbb{N}}^{\mathcal{T}_{\mathcal{B}}} \circ \tilde{f} \circ (\hat{h}_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}}})^{-1}$. Then f is clearly an isomorphism since $\hat{h}_{\mathbb{N}}^{\mathcal{T}_{\mathcal{B}}}$, $\tilde{f}$, and $\hat{h}_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}}}$ all are.

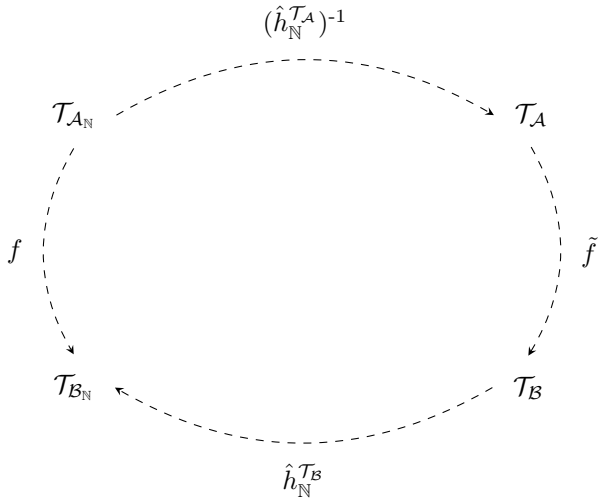


Figure 6.1: Construction of f from $\tilde{f}$ and $\hat{h}$ in proof of Lemma 6.12, that $\mathbf{d} \in \mathrm{DgSp}_{\mathcal{A}}(R) \implies \mathbf{d} \in \mathrm{DgSp}_{\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}}(\hat{R})$.

We note further one nice property of f and how it relates to g. Since $g(R)$ is simply some set of natural numbers in $\mathcal{B}$, we can think of $g(R)$ as itself a relation on $\mathcal{B}$. Using Equation 6.2 we can therefore define a corresponding relation $\widehat{g(R)}$ on $\mathcal{T}_{\mathcal{B}_{\mathbb{N}}}$. By the way we've defined f and $\tilde{f}$, this gives

us that $f(\hat{R}) = \widehat{g(R)}$:

$$\begin{aligned}
x \in f(\hat{R}) &\iff x = f\big(\hat{h}_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}}}([a]_{E_{n+1}})\big) \text{ for some } a \in R \quad \text{(by Equations 6.1 and 6.2)} \\
&\iff x = (\hat{h}_{\mathbb{N}}^{\mathcal{T}_{\mathcal{B}}} \circ \tilde{f} \circ (\hat{h}_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}}})^{-1}) \circ \hat{h}_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}}}([a]_{E_{n+1}}) \text{ for some } a \in R \\
&\iff x = (\hat{h}_{\mathbb{N}}^{\mathcal{T}_{\mathcal{B}}} \circ \tilde{f})([a]_{E_{n+1}}) \text{ for some } a \in R \\
&\iff x = \hat{h}_{\mathbb{N}}^{\mathcal{T}_{\mathcal{B}}}\big(\tilde{f}([a]_{E_{n+1}})\big) \text{ for some } a \in R \\
&\iff x = \hat{h}_{\mathbb{N}}^{\mathcal{T}_{\mathcal{B}}}\big(p(\mathcal{T}_{\mathcal{B}}, [g(a)]_{E^{\mathcal{B}}_{n+1}}, n+1)\big) \text{ for some } a \in R \\
&\iff x = \hat{h}_{\mathbb{N}}^{\mathcal{T}_{\mathcal{B}}}\big([g(a)]_{E^{\mathcal{B}}_{n+1}}\big) \text{ for some } a \in R \\
&\iff x \in \widehat{g(R)} \quad \text{(by Equation 6.2)}
\end{aligned}$$

We can now put it all together:

$$\begin{aligned}
\deg\big(f(\hat{R})\big) &= \deg\big(\widehat{g(R)}\big) \quad \text{(since } f(\hat{R}) = \widehat{g(R)}\text{)} \\
&= \deg\big(g(R)\big) \quad \text{(by Lemma 6.10)} \\
&= \mathbf{d} \quad \text{(by assumption)}
\end{aligned}$$

Therefore, we have found a tree, namely $\mathcal{T}_{\mathcal{B}_{\mathbb{N}}}$ which is isomorphic to $\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}$ via some isomorphism f, under which we have that $\deg\big(f(\hat{R})\big) = \mathbf{d}$. Therefore $\mathbf{d} \in \mathrm{DgSp}_{\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}}(\hat{R})$. □

Lemma 6.13. *Let $\mathcal{A} = (\mathbb{N}, E_1, \ldots, E_n)$ be a computable finitely nested equivalence structure, and let $\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}$ be its corresponding tree built as in Theorem 4.41. Furthermore, let R be some relation on $\mathcal{A}$ and let $\hat{R}$ be its corresponding relation on $\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}$ as defined in Equation 6.1. Then,*

$$DgSp_{\mathcal{A}}(R) \supseteq DgSp_{\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}}(\hat{R})$$

Proof. Let $\mathbf{d} \in \mathrm{DgSp}_{\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}}(\hat{R})$. This means there exists some computable tree, $\mathcal{S} = (\mathbb{N}, \prec_{\mathcal{S}})$ which is isomorphic to $\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}$ via an isomorphism f such that $\deg\big(f(\hat{R})\big) = \mathbf{d}$. Now to show that $\mathbf{d} \in \mathrm{DgSp}_{\mathcal{A}}(R)$, we need to show there exists a computable nested equivalence structure isomorphic to $\mathcal{A}$ via some isomorphism g such that $\deg\big(g(R)\big) = \mathbf{d}$. We will show that $\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}$ is such a nested equivalence structure.

First note that as constructed in Section 4.4, $\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}$ has domain $\mathbb{N}$ and is computable since $\mathcal{S}$ is computable. Additionally, as constructed in Section 5.1, $\mathcal{A}_{\mathcal{S}_{\mathbb{N}}} = F(\mathcal{S})$ where F is again our functor $F : \mathbf{FFT} \to \mathbf{NEquiv}$. By Corollary 5.15 and Theorem 5.10, we know that $\mathcal{A}_{\mathcal{S}_{\mathbb{N}}} \simeq \mathcal{A}$. Therefore, we

need only build an isomorphism g which yields a degree $\mathbf{d}$ image of the relation. Before building g, we will first define an isomorphism $\tilde{g} : \mathcal{T}_{\mathcal{A}} \to \mathcal{S}_{\mathbb{N}}$. An overview of what we will build is contained in Figure 6.2.

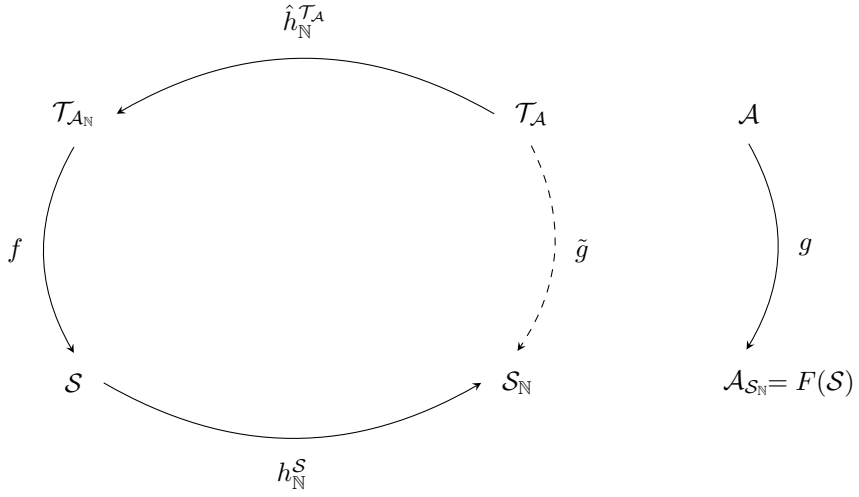


Figure 6.2: Construction of $\tilde{g}$ from f, $\hat{h}$, and h in proof of Lemma 6.13, that $\mathbf{d} \in \mathrm{DgSp}_{\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}}(\hat{R}) \implies \mathbf{d} \in \mathrm{DgSp}_{\mathcal{A}}(R)$.

Formally, we define $\tilde{g} = h_{\mathbb{N}}^{\mathcal{S}} \circ f \circ \hat{h}_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}}}$. Then $\tilde{g}$ is an isomorphism since f, $h_{\mathbb{N}}^{\mathcal{S}}$, and $\hat{h}_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}}}$ are. It is then easy to establish the following facts:

1. $(h_{\mathbb{N}}^{\mathcal{S}} \circ f)(\hat{R}) = \tilde{g}(\hat{R}^{\mathcal{T}_{\mathcal{A}}})$, and
2. $f(\hat{R}) \equiv_T (h_{\mathbb{N}}^{\mathcal{S}} \circ f)(\hat{R})$.

For 1, we relate $\hat{R}$ to $\hat{R}^{\mathcal{T}_{\mathcal{A}}}$ as in Equation 6.3 to get the following.

$$\begin{aligned}
x \in (h_{\mathbb{N}}^{\mathcal{S}} \circ f)(\hat{R}) &\iff x = h_{\mathbb{N}}^{\mathcal{S}}\big(f(j)\big) \quad \text{for some } j \in \hat{R} \\
&\iff x = h_{\mathbb{N}}^{\mathcal{S}}\big(f(\hat{h}_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}}}([a]_{E_{n+1}}))\big) \quad \text{for some } \hat{h}_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}}}([a]_{E_{n+1}}) \in \hat{R} \\
&\qquad \text{(since } \hat{h}_{\mathbb{N}}^{\mathcal{T}_{\mathcal{A}}} \text{ is an isomorphism, and therefore onto)} \\
&\iff x \in \tilde{g}(\hat{R}^{\mathcal{T}_{\mathcal{A}}})
\end{aligned}$$

For 2, we begin with $\geq_T$. Given some $[j]_{E_i^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}} \in \mathcal{S}_{\mathbb{N}}$ and an $f(\hat{R})$-oracle, we can determine whether $[j]_{E_i^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}} \in (h_{\mathbb{N}}^{\mathcal{S}} \circ f)(\hat{R})$ via the following process. Calculate $\mathrm{level}_{\mathcal{S}_{\mathbb{N}}}([j]_{E_i^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}})$. This is a computable process by Lemma 4.3. By Corollary 4.16, $i = \mathrm{level}_{\mathcal{S}_{\mathbb{N}}}([j]_{E_i^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}})$. So, check whether $\mathrm{level}_{\mathcal{S}_{\mathbb{N}}}([j]_{E_i^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}}) = n+1$. If $i \neq n+1$, then $[j]_{E_i^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}} \notin (h_{\mathbb{N}}^{\mathcal{S}} \circ f)(\hat{R})$. If $i = n+1$, then continue on and

compute $(h^{\mathcal{S}}_{\mathbb{N}})^{\text{-}1}([j]_{E_i^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}})$. This is a computable process since $h^{\mathcal{S}}_{\mathbb{N}}$ is computable, and by definition this gives us the jth end node of $\mathcal{S}$. Now, ask our $f(\hat{R})$-oracle whether $(h^{\mathcal{S}}_{\mathbb{N}})^{\text{-}1}([j]_{E_i^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}}) \in f(\hat{R})$. If yes, then $h^{\mathcal{S}}_{\mathbb{N}}\big((h^{\mathcal{S}}_{\mathbb{N}})^{\text{-}1}([j]_{E_i^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}})\big) = [j]_{E_i^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}} \in (h^{\mathcal{S}}_{\mathbb{N}} \circ f)(\hat{R})$. If no, then $[j]_{E_i^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}}$ *not* $\in (h^{\mathcal{S}}_{\mathbb{N}} \circ f)(\hat{R})$. Next, given some $x \in \mathcal{S}$ and an $(h^{\mathcal{S}}_{\mathbb{N}} \circ f)(\hat{R})$-oracle, we can determine whether $x \in f(\hat{R})$ via the following process. Calculate $\text{level}_{\mathcal{S}}(x)$, which is a computable process. If $\text{level}_{\mathcal{S}}(x) \neq n+1$, then $x \notin f(\hat{R})$. If $\text{level}_{\mathcal{S}}(x) = n+1$, then this means that x is some end node of $\mathcal{S}$. Then, enumerate end nodes of $\mathcal{S}$ in order (as in Lemma 4.8) until we have found j such that $x = e_j$. This means, then, that $h^{\mathcal{S}}_{\mathbb{N}}(x) = [j]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}}$. So, we now ask our $(h^{\mathcal{S}}_{\mathbb{N}} \circ f)(\hat{R})$-oracle whether $[j]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}} \in (h^{\mathcal{S}}_{\mathbb{N}} \circ f)(\hat{R})$. If yes, then $x \in f(\hat{R})$. If no, then $x \notin f(\hat{R})$.

Now we can define an isomorphism $g : \mathcal{A} \to \mathcal{A}_{\mathcal{S}_{\mathbb{N}}}$. We define g as follows:

$$g(a) = b \overset{\text{defn}}{\iff} \tilde{g}([a]_{E_{n+1}}) = [b]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}}$$

We first show that g is indeed 1-1. Let $g(a_1) = g(a_2) = b$. This implies that $\tilde{g}([a_1]_{E_{n+1}}) = \tilde{g}([a_2]_{E_{n+1}}) = [b]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}}$. But $\tilde{g}$ is itself an isomorphism, and therefore 1-1. Therefore $[a_1]_{E_{n+1}} = [a_2]_{E_{n+1}}$, and hence $a_1 = a_2$ (since E_{n+1} corresponds to the equivalence relation of equality).

Now to show that g is onto, we let $b \in \mathcal{A}_{\mathcal{S}_{\mathbb{N}}}$. This means that $[b]_{E_{n+1}} \in C_{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}$, and hence $[b]_{E_{n+1}}$ is a node of $\mathcal{S}_{\mathbb{N}}$. Therefore, since $\tilde{g}$ is onto, we know there must exist some node in $\mathcal{T}_{\mathcal{A}}$, call it x, such that $\tilde{g}(x) = [b]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}}$. Since $\tilde{g}$ preserves the order relation on trees and $[b]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}}$ is at level $n+1$ of $\mathcal{S}_{\mathbb{N}}$, we know that x must be at level $n+1$ of $\mathcal{T}_{\mathcal{A}}$. Since the nodes of $\mathcal{T}_{\mathcal{A}}$ are labelled syntactically like equivalence classes, this means that $x = [a]_{E_{n+1}}$ for some $a \in \mathcal{A}$. Therefore $\tilde{g}([a]_{E_{n+1}}) = \tilde{g}(x) = [b]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_{\mathbb{N}}}}}$. By the way we defined g, this gives us that $g(a) = b$, and hence g is indeed onto.

Finally, we show that g preserves our equivalence relations.

$$
\begin{aligned}
aE_ib &\iff [a]_{E_i} = [b]_{E_i} \text{ (as equivalence relations)} \\
&\iff [a]_{E_i} \succeq_{\mathcal{T}_\mathcal{A}} [c]_{E_i} \text{ and } [b]_{E_i} \succeq_{\mathcal{T}_\mathcal{A}} [c]_{E_i} \\
&\qquad \text{for } c \text{ the least element } E_i\text{-equivalent to } a \text{ and } b \\
&\qquad \text{(by the way we built } \mathcal{T}_\mathcal{A} \text{ in Section 4.5)} \\
&\iff \tilde{g}([a]_{E_{n+1}}) \succeq_{\mathcal{S}_\mathbb{N}} \tilde{g}([c]_{E_i}) \text{ and } \tilde{g}([b]_{E_{n+1}}) \succeq_{\mathcal{S}_\mathbb{N}} \tilde{g}([c]_{E_i}) \\
&\qquad \text{(since } g \text{ and isomorphism and therefore preserves order)} \\
&\iff [g(a)]_{E_{n+1}}) \succeq_{\mathcal{S}_\mathbb{N}} \tilde{g}([c]_{E_i}) \text{ and } [g(b)]_{E_{n+1}}) \succeq_{\mathcal{S}_\mathbb{N}} \tilde{g}([c]_{E_i}) \\
&\qquad \text{(by the way we defined } g) \\
&\iff \exists \text{ node } x = \tilde{g}([c]_{E_i}) \text{ at level } i \text{ s.t.} \\
&\qquad [g(a)]_{E_{n+1}}) \succeq_{\mathcal{S}_\mathbb{N}} x \text{ and } [g(b)]_{E_{n+1}}) \succeq_{\mathcal{S}_\mathbb{N}} x \\
&\iff g(a)E_i^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}} g(b) \quad \text{(by Equation 4.6)}
\end{aligned}
$$

Therefore as defined g is 1-1, onto, and preserves equivalence relations. Hence g is indeed an isomorphism.

Finally, we show that $g(R) \equiv_T \tilde{g}(\hat{R}^{\mathcal{T}_\mathcal{A}})$. We first note the following.

$$
\begin{aligned}
b \in g(R) &\iff b = g(a) \text{ for some } a \in R \\
&\iff \tilde{g}([a]_{E_{n+1}}) = [b]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}} \text{ for some } a \in R \\
&\iff [b]_{E_{n+1}^{\mathcal{A}_{\mathcal{S}_\mathbb{N}}}} \in \tilde{g}(\hat{R}^{\mathcal{T}_\mathcal{A}})
\end{aligned}
$$

Therefore, it is easy to see that given a $g(R)$-oracle we can easily compute $\tilde{g}(\hat{R}^{\mathcal{T}_\mathcal{A}})$. Similarly, given a $\tilde{g}(\hat{R}^{\mathcal{T}_\mathcal{A}})$-oracle we can easily compute $g(R)$.

We now put this together with facts 1. and 2. as established earlier in the proof:

$$\deg\big(g(R)\big) = \deg\big(\tilde{g}(\hat{R}^{\mathcal{T}_\mathcal{A}})\big) = \deg\big((h_\mathbb{N}^\mathcal{S} \circ f)(\hat{R})\big) = \deg\big(f(\hat{R})\big) = \mathbf{d}$$

Therefore, we found a computable nested equivalence structure $\mathcal{A}_{\mathcal{S}_\mathbb{N}}$ which is isomorphic to $\mathcal{A}$ via isomorphism g under which $\deg\big(g(R)\big) = \mathbf{d}$. This means that $\mathbf{d} \in \mathrm{DgSp}_\mathcal{A}(R)$. □

This completes the proof of Theorem 6.11. The Turing degree spectrum of a relation on a finitely

nested equivalence structure is identical to the Turing degree spectrum of the corresponding relation on the corresponding tree.

We can continue our examination of Turing degree spectra by exploring the notion of the least Turing degree. This concept was introduced by Richter in [40]. Richter proved the following result for trees which are represented via a partial order, the same definition we are using here.

Theorem 6.14 (Richter, [41])**.** *Let $\mathcal{T} = (T, \prec_{\mathcal{T}})$ be some countable tree which has no computable copy. Then DgSp($\mathcal{T}$) has no least degree.*

In particular, this means that any full finite height tree with no computable copy has no least degree in its Turing degree spectrum. Therefore, given any finitely nested equivalence structure $\mathcal{A}$, its corresponding tree $\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}$ (as built in Theorem 4.41) has no least degree in its Turing degree spectrum if the tree $\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}$ has no computable copy. In order to apply Richter's result to finitely nested equivalence structures, then, we already have most of the necessary framework in place. We need only prove the following easy lemma.

Lemma 6.15. *Let $\mathcal{A}$ be some finitely nested equivalence structure and let $\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}$ be its corresponding full finite height tree as in Theorem 4.41. Then $\mathcal{A}$ does not have a computable copy $\iff$ $\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}$ does not have a computable copy.*

Proof. This follows directly from Theorem 6.9. If $\mathcal{A}$ has no computable copy, this means that $\mathbf{0} \notin \mathrm{DgSp}(\mathcal{A})$. By Theorem 6.9, this in turn implies that $\mathbf{0} \notin \mathrm{DgSp}(\mathcal{T}_{\mathcal{A}_{\mathbb{N}}})$, and hence $\mathcal{T}_{\mathcal{A}_{\mathbb{N}}}$ has no computable copy. The other direction follows similarly.

□

We can now state the theorem.

Theorem 6.16. *Let $\mathcal{A}$ be a finitely nested equivalence structure which has no computable copy. Then DgSp($\mathcal{A}$) has no least degree.*

Proof. We combine the results in Theorem 6.9, Lemma 6.15, and Theorem 6.14. □

6.4 Future Research

One simple course of future research may be to come up with a nicer characterization for trees of "finite type". Though the current characterization is complete, there may be a simpler way to view the concept when thought of as nested equivalence structures.

Another avenue of future research is to see if we can use the language of nested equivalence structures to perhaps take a different approach to examining some of the current open problems for finite height trees. Although any such results obtained would only apply to full finite height trees, the endeavor may still yield some interesting outcomes. Historically difficult problems on trees may become more straightforward when examined on nested equivalence structures.

References

[1] Uri Andrews, Steffen Lempp, Joseph S. Miller, Keng Meng Ng, Luca San Mauro, and Andrea Sorbi. Universal computably enumerable equivalence relations. *Journal of Symbolic Logic*, 79(1):60–88, 2014.

[2] C. J. Ash. Categoricity in hyperarithmetical degrees. *Annals of Pure and Applied Logic*, 34(1):1–14, 1987.

[3] C. J. Ash and J. F. Knight. *Computable Structures and the Hyperarithmetical Hierarchy*, volume 144 of *Studies in Logic and the Foundations of Mathematics*. Elsevier, 2000.

[4] S. Awodey. *Category Theory*. Number 52 in Oxford Logic Guides. Oxford University Press, second edition, 2010.

[5] W. Calvert, V.S. Harizanov, J.F. Knight, and S. Miller. Index sets of computable structures. *Algebra and Logic*, 45(5):306–325, 2006.

[6] Wesley Calvert, Douglas Cenzer, Valentina Harizanov, and Andrei Morozov. Effective categoricity of equivalence structures. *Annals of Pure and Applied Logic*, 141(1-2):61–78, 2006.

[7] D. Cenzer, V. Harizanov, and J. B. Remmel. Computability-theoretic properties of injection structures. *Algebra and Logic*, 53(1):39–69, 2014.

[8] Douglas Cenzer, Valentina Harizanov, and Jeffrey B. Remmel. Σ^0_1 and Π^0_1 equivalence structures. In *Mathematical theory and computational practice*, volume 5635 of *Lecture Notes in Computer Science*, pages 99–108. Springer, Berlin, 2009.

[9] Douglas Cenzer, Valentina Harizanov, and Jeffrey B. Remmel. Effective categoricity of injection structures. In *Models of computation in context*, volume 6735 of *Lecture Notes in Computer Science*, pages 51–60. Springer, Heidelberg, 2011.

[10] Douglas Cenzer, Valentina Harizanov, and Jeffrey B. Remmel. Σ_1^0 and Π_1^0 equivalence structures. *Annals of Pure and Applied Logic*, 162(7):490–503, 2011.

[11] Douglas Cenzer, Valentina Harizanov, and Jeffrey B. Remmel. Two-to-one structures. *Journal of Logic and Computation*, 23(6):1195–1223, 2013.

[12] Douglas Cenzer, Geoffrey Laforte, and Jeffrey Remmel. Equivalence structures and isomorphisms in the difference hierarchy. *Journal of Symbolic Logic*, 74(2):535–556, 2009.

[13] Jennifer Chubb, Andrey Frolov, and Valentina Harizanov. Degree spectra of the successor relation of computable linear orderings. *Archive for Mathematical Logic*, 48(1):7–13, 2009.

[14] S. Barry Cooper. *Computability Theory.* Chapman Hall/CRC Mathematics Series. Chapman and Hall/CRC, 2004.

[15] N. J. Cutland. *Computability.* Cambridge University Press, 1980.

[16] Ekaterina B. Fokina and Sy-David Friedman. Equivalence relations on classes of computable structures. In Klaus Ambos-Spies, Benedikt Löwe, and Wolfgang Merkle, editors, *Mathematical Theory and Computational Practice*, volume 5635 of *Lecture Notes in Computer Science*, pages 198–207. Springer Berlin Heidelberg, 2009.

[17] Ekaterina B. Fokina, Sy-David Friedman, Valentina Harizanov, Julia F. Knight, Charles McCoy, and Antonio Montalbán. Isomorphism relations on computable structures. *Journal of Symbolic Logic*, 77(1):122–132, 2012.

[18] Ekaterina B. Fokina, Valentina Harizanov, and Alexander Melnikov. Computable model theory. In Rod Downey, editor, *Turing's Legacy : Developments from Turing's Ideas in Logic*, volume 42 of *Lecture Notes in Logic*, pages 124–194. Cambridge University Press, Cambridge, May 2014.

[19] Ekaterina B. Fokina, Iskander Kalimullin, and Russell Miller. Degrees of categoricity of computable structures. *Archive for Mathematical Logic*, 49(1):51–67, 2010.

[20] Sergey Goncharov, Valentina Harizanov, Julia Knight, Charles McCoy, Russell Miller, and Reed Solomon. Enumerations in computable structure theory. *Annals of Pure and Applied Logic*, 136(3):219–246, 2005.

[21] Sergey S. Goncharov and Bakhadyr Khoussainov. Complexity of computable models. CDMTCS Research Report Series 190, Centre for Discrete Mathematics and Theoretical Computer Science, University of Auckland, 2002.

[22] S.S. Goncharov. Autostability and computable families of constructivizations. *Algebra and Logic*, 14(6):392–409, 1975.

[23] Valentina S. Harizanov. *Degree spectrum of a recursive relation on a recursive structure*. PhD thesis, University of Wisconsin-Madison, 1987.

[24] Valentina S. Harizanov. Pure computable model theory. In Yu. L. Ershov, S. S. Goncharov, A. Nerode, and J. B. Remmel, editors, *Handbook of recursive mathematics, Vol. 1 — Recursive Model Theory*, volume 138 of *Stud. Logic Found. Math.*, pages 3–114. North-Holland, Amsterdam, 1998.

[25] Valentina S. Harizanov. Computability-theoretic complexity of countable structures. *The Bulletin of Symbolic Logic*, 8(4):457–477, December 2002.

[26] M. Harrison-Trainor, A. Melnikov, R. Miller, and A. Montalbàn. Computable functors and effective interpretability, to appear.

[27] Denis R. Hirschfeldt. Degree spectra of relations on computable structures. *Bulletin of Symbolic Logic*, 6(2):197–212, 2000.

[28] Denis R. Hirschfeldt, Bakhadyr Khoussainov, Richard A. Shore, and Arkadii M. Slinko. Degree spectra and computable dimensions in algebraic structures. *Annals of Pure and Applied Logic*, 115(1-3):71–113, 2002.

[29] Asher M. Kach and Daniel Turetsky. Δ^0_2-categoricity of equivalence structures. *New Zealand Journal of Mathematics*, 39:143–149, 2009.

[30] Bakhadyr Khoussainov and Richard A. Shore. Computable isomorphisms, degree spectra of relations, and Scott families. *Annals of Pure and Applied Logic*, 93(1-3):153–193, 1998. Computability theory.

[31] Steffen Lempp, Charles McCoy, Russell Miller, and Reed Solomon. Computable categoricity of trees of finite height. *Journal of Symbolic Logic*, 70(1):151–215, 2005.

[32] Saunders Mac Lane. *Categories for the Working Mathematician*. Graduate Texts in Mathematics. Springer, second edition, 1998.

[33] David Marker. Non Σ_n axiomatizable almost strongly minimal theories. *The Journal of Symbolic Logic*, 54(3):921–927, 1989.

[34] Terrence S. Millar. Pure recursive model theory. In *Handbook of computability theory*, volume 140 of *Studies in logic and the foundation of mathematics*, pages 507–532. North-Holland, Amsterdam, 1999.

[35] R. Miller, J. Park, B. Poonen, H. Schoutens, and A. Shlapentokh. A computable functor from graphs to fields, to appear.

[36] Russell Miller. The computable dimension of trees of infinite height. *Journal of Symbolic Logic*, 70(1):111–141, 2005.

[37] Antonio Montalbàn. Computability theoretic classifications for classes of structures. In *Proceedings of the International Congress of Mathematicians*, volume II, pages 79–101, 2014.

[38] Victor A. Ocasio. *Computability in the class of Real Closed Fields*. PhD thesis, University of Notre Dame, 2014.

[39] J. B. Remmel. Recursively categorical linear orderings. *Proceedings of the American Mathematical Society*, 83(2):387–391, 1981.

[40] Linda Jean Richter. *Degrees of unsolvability of models*. PhD thesis, University of Illinois at Urbana-Champaign, 1977.

[41] Linda Jean Richter. Degrees of structures. *The Journal of Symbolic Logic*, 46(4):723–731, 1981.

[42] R. I. Soare. *Recursively Enumerable Sets and Degrees. A Study of Computable Functions and Computably Generated Sets*. Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1987.

[43] R. I. Soare. Computability theory and applications. Draft 502, in preparation, January 2008.

Index