PROCEEDINGS OF THE
AMERICAN MATHEMATICAL SOCIETY
Volume 142, Number 5, May 2014, Pages 1795–1811
S 0002-9939(2014)11878-8
Article electronically published on February 4, 2014

TURING COMPUTABLE EMBEDDINGS OF EQUIVALENCES OTHER THAN ISOMORPHISM

MATTHEW WRIGHT

(Communicated by Julia Knight)

Abstract. The concept of a Turing computable embedding was introduced by Calvert, Cummins, Miller, and Knight as an effective analog of Borel embeddings. However, while the theory of Borel embeddings is often applied to general equivalence relations, research so far in Turing computable embeddings has focused mostly on the isomorphism relation. This paper investigates Turing computable embeddings of general equivalence relations. We first examine a relation on trees which is coarser than isomorphism and can be handled in much the same way as isomorphism. We then examine the computable isomorphism relation on computable structures, a finer relation which requires new techniques. The main result of this paper is that every class of computable structures under computable isomorphism embeds into the class of computable copies of ω as a linear ordering under the computable isomorphism, indicating that Turing computable embeddings are unlikely to produce meaningful distinctions when restricted to computable isomorphism relations.

1. Introduction

A great deal of research in logic—especially descriptive set theory and computability theory—deals with characterizing how difficult various problems are. One approach when it comes to comparing equivalence relations on classes $\mathcal{A}$ and $\mathcal{B}$ of structures is to give some function

$$f : \mathcal{A} \to \mathcal{B}$$

that respects the relations; that way, if we have some way of deciding whether two members of $\mathcal{B}$ are related, we can do the same for $\mathcal{A}$. In this sense, the relation on $\mathcal{B}$ is "at least as hard" as that on $\mathcal{A}$.

The theory of Borel reductions, introduced by Friedman and Stanley [5], deals with the case where f is restricted to be a Borel function, and it has been very successful. Borel reducibility is able to make distinctions between many equivalence relations that intuitively should be distinct, and many relations we see in practice are equivalent to one of a few very natural equivalence relations. Furthermore, several interesting structure theorems have been proved about Borel equivalence relations (see Hjorth [6] and Kanovei [7] for details).

Calvert, Cummins, Miller, and Knight introduced the concept of a Turing computable embedding, or *tc*-embedding, in [1] as an effective analog to the concept of

Received by the editors February 28, 2011 and, in revised form, November 28, 2011; April 27, 2012; and May 29, 2012.

2010 *Mathematics Subject Classification.* Primary 03D45; Secondary 03C57.

a Borel embedding. The formal definition will be given later, but the main difference between a tc-embedding and a Borel embedding is that f is now taken to be a Turing computable functional rather than a Borel function. Turing computable embeddings make finer distinctions than Borel embeddings while still preserving a lot of their useful features. Furthermore, tc-embeddability can often be characterized syntactically, by the complexity of formulas (see the Pull-back Theorem, below), and are very closely related to the computational difficulty of invariants of structures. For example, the isomorphism relations on PF (the class of prime fields), FLO (the class of finite linear orderings), FVS (the class of finite vector spaces), VS (the class of vector spaces), and LO (the class of linear orderings) can be classified according to tc reductions; in fact, it has been shown in [1] in that

$$PF <_{tc} FLO <_{tc} FVS <_{tc} VS <_{tc} LO,$$

where no two of the classes are tc-equivalent. From a computability perspective, these reductions correspond to the fact that computing a set of invariants for the classes gets harder as we move toward the right. In the setting of Borel reductions, all of these classes but the last are equivalent.

Borel embeddings have been extensively studied in settings other than isomorphism. In particular, in [2] Camerlo showed that computable equivalence relations on several structures are universal countable Borel relations. In the setting of tc-embeddings, there is work by Fokina and Friedman in [3] and Fokina, Friedman, and Törnquist in [4], on Σ^1_1 equivalence relations. However, so far there has been less research in how tc-embeddings behave with other equivalence relations, including arithmetic relations. We will examine classes of structures under other equivalence relations. Section 3 deals with an equivalence relation on trees which is coarser than isomorphism and shows how it can be classified using the same techniques that were used to classify structures like vector spaces and that it is equivalent to a class under isomorphism, demonstrating that other equivalences can fit in very well alongside isomorphism relations.

In sections 4 and 5 we examine the relation of computable isomorphism—because tc-embeddings are an effectivization of Borel embeddings, it seems reasonable to examine them in the context of effective isomorphism. Unfortunately, when we restrict ourselves to the setting of computable isomorphism, tc-embeddings do not seem to give useful distinctions as we might expect them to. In fact, as we show in Theorem 5.2, even simple classes such as the class of computable copies of ω as an ordering are complete for tc-embeddings of computable isomorphism relations; that is, *any* class of uniformly computable structures, under computable isomorphism, embeds into ω. In addition to showing that a seemingly natural setting for tc-embeddings fails, the result also provides a construction of an embedding based on guessing at the relation itself rather than on invariants of individual structures, which therefore reflects only the relations between structures instead of the structures themselves, unlike most existing constructions.

2. Basic definitions and previous results

We will use the notation from [11] for computability theory. The eth partial computable function is denoted $\varphi_e(x)$, and Φ_e^X denotes the eth Turing functional.

Before we can talk about computable maps of structures, we need to be clear on exactly what kinds of structures we are working with and how we represent them.

In model theory we typically allow any underlying set (universe) for structures; in the computable setting, though, all structures must be countable, and so we may assume without loss of generality that all infinite structures have universe ω. A constant is therefore just a natural number, an n-ary relation is a subset of ω^n, and an n-ary function is a function from ω^n to ω.

Definition 2.1. Let S be a structure. The *diagram* of S is the set of all quantifier-free first-order statements in the language of S which are true in S. The diagram can be thought of as a subset of ω by identifying each formula with its Gödel number. By a *presentation* or *copy* of a structure we mean a diagram of an isomorphic structure.

From now on we will identify a structure with its diagram, so there will not be any special notation for the diagram of a structure. When we say "S is computable", for example, we really mean "the diagram of S is a computable subset of ω".

Definition 2.2 (Turing computable embedding; Calvert, Cummins, Miller, Knight [1])**.** Let $\mathcal{L}_\mathcal{A}$ and $\mathcal{L}_\mathcal{B}$ be languages, and let $\mathcal{A}$ be a class of $\mathcal{L}_\mathcal{A}$-structures and $\mathcal{B}$ a class of $\mathcal{L}_\mathcal{B}$-structures. A *tc-embedding* from $\mathcal{A}$ to $\mathcal{B}$ is a Turing functional $\Phi_e : \mathcal{A} \to \mathcal{B}$ such that:

(1) For any $A \in \mathcal{A}$, we have $\Phi_e^A \in \mathcal{B}$.
(2) Φ_e respects isomorphism; that is, $A_1 \cong A_2 \iff \Phi_e^{A_1} \cong \Phi_e^{A_2}$.

If a *tc*-embedding exists from $\mathcal{A}$ to $\mathcal{B}$, we write $\mathcal{A} \leq_{tc} \mathcal{B}$.

Intuitively, a *tc*-embedding from one class of structures $\mathcal{A}$ to another class $\mathcal{B}$ is a computable map from structures in $\mathcal{A}$ to structures in $\mathcal{B}$ that preserves the isomorphism relation. If $\mathcal{A} \leq_{tc} \mathcal{B}$ and we have some way to solve the isomorphism problem in $\mathcal{B}$, we can solve the isomorphism problem in $\mathcal{A}$. It is important to note that the embedding does not map *elements* of structures in $\mathcal{A}$; it is a map on the structures themselves.

The Pull-back Theorem, due to Knight, Miller, and Vanden Boom [8], relates *tc*-embeddability of classes of structures to how hard it is to define invariants for those structures with formulas in $L^c_{\omega_1\omega}$, the system of computable infinitary logic, which is defined as follows:

Definition 2.3. $L^c_{\omega_1\omega}$ is the smallest system of formulas such that:

(1) Any formula θ in the language of first-order logic is a formula of $L^c_{\omega_1\omega}$.
(2) If $\{\theta_i\}_{i\in\omega}$ is a c.e. set of formulas in $L^c_{\omega_1\omega}$, then their *infinitary meet*, written

$$\bigwedge\nolimits_{i\in\omega} \theta_i,$$

and their *infinitary join*

$$\bigvee\nolimits_{i\in\omega} \theta_i$$

are both in $L^c_{\omega_1\omega}$.

If $\theta \in L^c_{\omega_1\omega}$, we say:

(1) θ is Σ_0 and Π_0 if it has no quantifiers or infinitary connectives;
(2) θ is Σ_{n+1} if

$$\theta \equiv \bigvee\nolimits_{i\in\omega} \exists \vec{x}_i \, \psi_i(\vec{x}_i),$$

where $\{\psi_i\}$ is a uniformly computable sequence of Π_n formulas;

(3) θ is Π_{n+1} if

$$\theta \equiv \bigwedge_{i\in\omega} \forall \vec{x}_i \, \psi_i(\vec{x}_i),$$

where $\{\psi_i\}$ is a uniformly computable sequence of Σ_n formulas.

Note that not all formulas are Π_n or Σ_n for some finite n.

In this paper, all infinitary formulas will be computable.

Remark 2.4. The hierarchy of formulas in $L^c_{\omega_1\omega}$ corresponds closely to the arithmetic hierarchy; the problem of deciding the truth of a Π_n (or Σ_n) formula in a computable structure is itself Π_n (or Σ_n). Using representation theorems for Σ_n predicates (see [10], [11]), we can come up with "guessing strategies" for Σ_n formulas. In particular, for any Σ_2 predicate $P(x)$, there is a computable function $f(x, s)$ taking only the values 0 and 1 so that if we let $f_x(s) = f(x, s)$, then

$$P(x) \iff \liminf f_x(s) = 1.$$

In other words, if $P(x)$ is true, then $f_x = 1$ in all but finitely many places, and if $P(x)$ is false, then $f_x = 0$ infinitely often.

We are now ready to state the Pull-back Theorem, a useful tool in working with *tc*-embeddings:

Theorem 2.5 (The Pull-back Theorem; Knight, Miller, and Vanden Boom [8])**.** *Let $\mathcal{A}$ be a class of structures in some language $\mathcal{L}_\mathcal{A}$ and $\mathcal{B}$ a class of structures in a language $\mathcal{L}_\mathcal{B}$. Let $f : \mathcal{A} \to \mathcal{B}$ be a tc-embedding, and let θ be a computable infinitary Σ_n formula in the language $\mathcal{L}_\mathcal{B}$. Then there is a computable Σ_n formula θ^* in the language $\mathcal{L}_\mathcal{A}$, called the* pull-back *of θ, such that $A \models \theta^* \iff f(A) \models \theta$.*

Remark 2.6. The Pull-back Theorem implies the same result with Π_n in place of Σ_n, by taking negations.

Using this theorem, it is often possible to precisely characterize which structures embed into a given class. For example, Quinn proved the following theorem.

Theorem 2.7 (Quinn [9])**.** *Let E_{fin} be the class of equivalence structures, all of whose equivalence classes are finite. Let $\mathcal{A}$ be a class of structures, all in the same language. Then $\mathcal{A} \leq_{tc} E_{fin}$ if and only if there is a computable sequence $\{\theta_i\}_{i\in\omega}$ of computable infinitary Σ_3 formulas such that if $A_1 \not\cong A_2$, then there is some i such that exactly one of $A_1 \models \theta_i$ and $A_2 \models \theta_i$ holds.*

However, less research has been done with equivalence relations on structures other than isomorphism, even though the definition of a *tc*-embedding still makes sense for any equivalence relation (and indeed, any relation at all). From here on, *tc*-embeddings will not necessarily deal with the isomorphism relation; we will always be explicit about which relation we are considering on the structures we deal with. The formal definition we will work with now is:

Definition 2.8. Let $\mathcal{L}_\mathcal{A}$ and $\mathcal{L}_\mathcal{B}$ be languages, and let $E_\mathcal{A}$ be an equivalence relation on a class $\mathcal{A}$ of $\mathcal{L}_\mathcal{A}$-structures, and $E_\mathcal{B}$ an equivalence relation on a class $\mathcal{B}$ of $\mathcal{L}_\mathcal{B}$-structures. A *tc-embedding* from $\mathcal{A}$ to $\mathcal{B}$ is a Turing functional $\Phi_e : \mathcal{A} \to \mathcal{B}$ such that:

(1) For any $A \in \mathcal{A}$, we have $\Phi_e^A \in \mathcal{B}$.
(2) Φ_e respects the equivalence relations; that is, $A_1 E_\mathcal{A} A_2$ iff $\Phi_e^{A_1} E_\mathcal{B} \Phi_e^{A_2}$.

To accommodate this more general setting, from now on we will write $\leq_{tc}$ as being between two relations rather than two classes of structures. When there may be confusion about the underlying structure, we will write it as a subscript.

Example 2.9. Under this notation, $\cong_{E_{fin}}$ is the isomorphism relation on E_{fin}.

Theorem 2.7 can easily be adapted to work in this more general setting. The following theorem goes through with exactly the same proof.

Theorem 2.10 (Quinn)**.** *Let $\mathcal{A}$ be a class of structures, all in the same language, under an equivalence relation $E_{\mathcal{A}}$. Then $(E_{\mathcal{A}}) \leq_{tc} (\cong_{E_{fin}})$ if and only if there is a computable sequence $\{\theta_i\}_{i\in\omega}$ of computable infinitary Σ_3 formulas such that for all $A_1, A_2 \in \mathcal{A}$,*

(1) *if $\neg(A_1 E_{\mathcal{A}} A_2)$, then there is some i such that exactly one of $A_1 \models \theta_i$ and $A_2 \models \theta_i$ holds, and*
(2) *if $A_1 E_{\mathcal{A}} A_2$, then for all $i \in \omega$*

$$A_1 \models \theta_i \quad \Longleftrightarrow \quad A_2 \models \theta_i.$$

3. Tree paths

So far we have looked only at examples of *tc*-embeddings of the isomorphism relations on classes of structures, the setting in which most research about *tc*-embeddings has been done. We will now start to examine other relations, starting with the relation on labeled binary trees having the same labeled infinite paths (a coarser relation than isomorphism). We will show that this relation is *tc*-equivalent to the isomorphism relation on E_{fin}. Let $\mathcal{T}$ be the class of binary trees, each considered as a structure in the language (r, R, L) where r is a constant symbol (representing the root node of the tree) and R and L are binary relations such that aRb if b immediately extends a on the right, and aLb if b immediately extends it on the left. To make it easier to work with trees, we label each node with a binary string as follows:

(1) r has the label ϵ, the empty string;
(2) if a has label σ and aLb, then we assign to b the label $\sigma 0$;
(3) if a has label σ and aRb, then we assign to b the label $\sigma 1$.

It should be clear that the set of labels is exactly what we would have if the tree were a tree in the "usual" sense, that is, a set of binary strings closed under prefixes. From now on we will not distinguish between an element and its label; for example, when $\sigma \in 2^{<\omega}$ the notation "$\sigma \in T$" means that there is a node in T with label σ.

Remark 3.1. Although we are labeling nodes with binary strings, it is important to keep in mind that these trees are *not* themselves sets of binary strings. Given the diagram of a tree T, there is no way to computably tell whether a node with a given label is in T (the set of labels of nodes in T is c.e. but not computable). In particular, if we are *building* the diagram of a tree, we may wait as long as we want before (if ever) adding children to a given node; if trees were given as sets of strings we would eventually have to commit to the number of children each node has.

Definition 3.2. Given a tree $T \in \mathcal{T}$, we define the *set of paths through* T, denoted $[T]$, as the set of all $x \in 2^\omega$ such that $(x \upharpoonright n) \in T$ for all $n \in \omega$. Finally, we define the equivalence relation $\simeq^p$ (where the superscript p is for "paths") by $T_1 \simeq^p T_2$ if $[T_1] = [T_2]$.

Our task now is to classify what structures embed into $\mathcal{T}$ under the relation $\simeq^p$. The following theorem shows that even though $\simeq^p$ is not the isomorphism relation, it ends up being equivalent to a class under isomorphism, indicating that even more general equivalence relations can sometimes fit into existing tc-equivalence classes. In this case, $\simeq^p$, just like E_{fin} and many other classes, is characterized by Σ_3 invariants.

Theorem 3.3. $\left(\cong_{E_{fin}}\right) \equiv_{tc} (\simeq^p)$.

Proof. We first show that $\left(\cong_{E_{fin}}\right) \leq_{tc} (\simeq^p)$.

We will start by constructing a tc-embedding from E_{fin} to $\simeq^p$. By Proposition 3.2.1 of [9], there is a computable sequence of computable infinitary Σ_2 formulas $\{\theta_i\}_{i\in\omega}$ so that for $E_1, E_2 \in E_{fin}$ not isomorphic, there is some n so that θ_n is true in exactly one of E_1, E_2; for example, formulas satisfying

$$E \models \theta_{\langle a,b\rangle} \iff E \text{ has at least } a \text{ many classes of size } b$$

will do.

We will work with these formulas to construct our embedding.

Our strategy follows the usual technique for constructing a tc-embedding: we will guess at the truth values of each θ_i, building a tree based on our current guesses. This process will ultimately code the value of each θ_i into the paths of our tree.

The tree T we construct will have the following properties:

(1) $0^i \in T$ for all $i \in \omega$ — that is, the path of all zeros is in $[T]$.
(2) If θ_n is false, $0^n1^i \in T$ for all $i \in \omega$; if θ_n is true, at least some string of the form 0^n1^i is not in T.
(3) Every element of T has the form 0^i1^j, with $i, j \in \omega$.

This tree consists of an infinite path on the left, with branches coming out on the right which are infinite only when the corresponding formula is true.

The construction of T proceeds in stages. The guessing strategy for Σ_2 formulas, mentioned in Remark 2.4, lets us assume that if θ_n is true there will be some stage after which we always guess that it is true, and if it is false we will infinitely often guess that it is false.

(1) Stage 0: assign 0 to be the root node.
(2) Stage s: assign some element to correspond to the string 0^s. For each $n < s$ such that we guess that θ_i is false at this stage, let i_n be minimal such that $0^n1^{i_n}$ is on the tree so far, and add an element corresponding to $0^n1^{i_n+1}$.

This completes the construction. If θ_n is false, there will be infinitely many stages in which we guess that it is false, and so there will be elements of the form 0^n1^i for all $i \in \omega$, resulting in an element of $[T]$. If on the other hand θ_n is true, there will be a maximal such i, and so there will be no contribution to $[T]$. Therefore, $[T]$ encodes exactly which θ_n are true; that is, two trees constructed this way are equivalent if and only if the same formulas were true in each construction.

We now turn to showing that $(\simeq^p) \leq_{tc} \left(\cong_{E_{fin}}\right)$.

By Theorem 2.10, it suffices to produce a series of Σ_2 formulas $\{\theta_n\}_{n\in\omega}$ so that if $T_1 \not\simeq^p T_2$, there is some n such that exactly one of $T_1 \models \theta_n$ and $T_2 \models \theta_n$ is true, and both have the same truth value when $T_1 \simeq^p T_2$. We will define them as follows: for natural numbers y and k, where y is treated as a node on a tree, let $\psi_{y,k}$ be the

Π_1 formula stating that the node y cannot be extended by k places, defined as

$$\psi_{y,k} \equiv \neg\exists z_1, \ldots, z_{k+1} \left(z_1 = y \wedge \left(\bigwedge_j z_j R z_{j+1} \vee z_j L z_{j+1} \right) \right).$$

Using $\psi_{y,j}$ we can define, for $t \in 2^{<\omega}$, a Σ_2 formula

$$\theta_t \equiv \exists z_1, \ldots, z_{|t|+1} \bigwedge_{\{i:t(i)=0\}} z_i L z_{i+1} \wedge \bigwedge_{\{i:t(i)=1\}} z_i R z_{i+1} \wedge \bigvee_n \psi_{z_{|t|+1},n}$$

which is true if and only if the node given by the string t cannot be extended to an infinite path; that is,

$$T \models \theta_t \iff \neg\exists x \in [T]\, (t \prec x).$$

Therefore, the set

$$S := \{t : T \models \theta_t\}$$

lets us find $[T]$ even without knowing T, and similarly from $[T]$ we can find S. Therefore, the set $\{\theta_t\}_{t\in 2^{<\omega}}$ works as our set of Σ_2 formulas, and so $(\simeq^p) \leq_{tc} (E_{fin})$ by Theorem 2.10. □

Remark 3.4. A similar (though less interesting) argument shows that trees in a language where children are given by functions instead of relations (which correspond more closely to trees as sets of strings closed under prefix) have embeddability characterized by Σ_1 relations instead: the formulas θ_t are Σ_1 in this language, and it is easy to construct a tree encoding the truth of Σ_1 predicates in a way similar to what we did with Σ_2 predicates.

4. Computable isomorphism

In the previous section we dealt with a relation on structures that was coarser than isomorphism: any isomorphic structures were equivalent, but two equivalent structures might not be isomorphic. In this section we will deal with the computable isomorphism relation, which is much *finer* than isomorphism; in fact, we will deal mostly with classes of structures *all of which* are classically isomorphic.

Definition 4.1. Computable presentations A and B of a structure are *computably isomorphic*, which we will write as $A \cong^c B$, if there is an isomorphism $f : A \to B$ which is computable. As before, when the structure is important we will include it as a subscript: if $\mathcal{A}$ is a class of structures, $\cong^c_{\mathcal{A}}$ is the computable isomorphism relation on $\mathcal{A}$.

We will examine classes consisting of all computable copies of a structure under computable isomorphism. Since *all* structures in such a class are classically isomorphic, the Pull-back Theorem is no longer of any use—a formula is either true in all of the structures or false in all of them; no set of formulas can separate the computable isomorphism classes.

The main result of the following section is Theorem 5.2, which shows that *every* computable isomorphism relation embeds into the computable isomorphism relation on copies of ω as a linear ordering. There are several new difficulties in this setting. Before, when we had the Pull-back Theorem, classifying structures tended to consist of first providing a way to code formulas of a certain complexity in the structures and then using the Pull-back Theorem to show that no structures requiring harder invariants could embed. When we showed that $\left(\cong_{E_{fin}}\right) \leq_{tc} (\simeq^p)$,

for example, we built a tree that encoded information about how many equivalence classes of each size appeared in the equivalence structure we were given; that information was enough to determine its isomorphism type. In the setting of a single isomorphism class and the equivalence relation of computable isomorphism, though, there are not in general useful invariants that we can guess at. To get around this problem, we will do a construction based on guessing whether *pairs* of structures are equivalent. In other words, rather than guessing at invariants in individual structures and encoding them, we will guess at the actual equivalence relation on pairs of structures.

But first, to help us get a feel for *tc*-embeddings in the context of computable isomorphism, we will go over a simple example. Here, and from now on, we will identify each ordinal with the class of computable copies of that ordinal (as an ordering).

Theorem 4.2. *Let $\alpha < \beta < \omega_1^{CK}$ be infinite ordinals. Then $(\cong_\alpha^c) \leq_{tc} (\cong_\beta^c)$.*

Proof. Since $\alpha < \beta$, we can write $\beta = \alpha + \gamma$ for some computable ordinal γ. Pick a specific computable copy C of γ. Given a copy of α, build a copy of $\alpha + \gamma$ by coding our copy of α with the even numbers and including a copy of C in the odd numbers.

It is clear that this produces a *tc*-embedding: if two copies of α are computably isomorphic by some function f, the function

$$g(x) = \begin{cases} 2f(x/2) & \text{if } x \text{ is even,} \\ x & \text{if } x \text{ is odd} \end{cases}$$

is a computable isomorphism between their images. Similarly, given a computable isomorphism f between two copies of $\alpha + \gamma$ in the image of our embedding,

$$g(x) = f(2x)/2$$

is a computable isomorphism between the original copies of α. □

Theorem 4.3. *For any infinite $\alpha < \omega_1^{CK}$, we have $(\cong_{\alpha+1}^c) \leq_{tc} (\cong_\alpha^c)$.*

Proof (sketch). We will build a *tc*-embedding by guessing at the last element of $\alpha + 1$. If we already knew which element was the last one, we could move it to the front of our copy of α while leaving all of the other elements alone. Since this copy of α differs only finitely from the original copy of $\alpha + 1$, the function between the two is computable, and so computable isomorphism is certainly preserved by this embedding.

Unfortunately, though, we *do not* know the last element of the copy of $\alpha + 1$, so we need a guessing strategy. If at some stage we believe a to be the last element, then in the copy of α we are building we will move a to the front. If we ever discover an element bigger than a, we start over, placing *that* element at the front of our copy of α and placing everything else, in its original order, to the right of any elements we had added to the copy of α while we still believed a to be the largest element of the copy of $\alpha + 1$.

At some stage we will come across the true largest element. We will have made only finitely many mistakes up to that point, and so the copy of α we build *still* differs only finitely from the copy of $\alpha + 1$. This embedding, then, once again respects computable isomorphism. □

5. *tc*-EMBEDDINGS BASED ON GUESSING AT RELATIONS

The above results give us hope that *tc*-embeddings in the context of computable isomorphism could in general be built similarly to the way they were built in the context of classical isomorphism: that is, by guessing at certain important information and building a structure based on those guesses. However, trying to go any further than the previous result based on this strategy seems to be a dead end. Consider trying to embed $\omega \times 2$ into ω. Even if we somehow could guess at which element was the limit element of $\omega \times 2$ (and it is not clear that we could), there is no obvious place to send it—there is no obvious way to "interleave" the two copies of ω in $\omega \times 2$. But surprisingly enough, $\omega \times 2$ *does* embed into ω; in fact, we have the following result, which we will prove later in this section:

Definition 5.1. An equivalence relation F on a class of uniformly partially computable structures $\mathcal{A} = \{A_0, A_1, \ldots\}$ is Σ_n (Π_n) if the relation $aEb \cong A_a F A_b$ is Σ_n (Π_n) in the usual sense.

Theorem 5.2. *Let $\mathcal{A} = \{A_0, A_1, \ldots\}$ be a class of uniformly partial computable structures, all in the same language, and let E be a Σ_3 equivalence relation on $\mathcal{A}$. Then $E \leq_{tc} (\cong^c_\omega)$.*

Remark 5.3. Because computable isomorphism relations are themselves Σ_3, we can apply Theorem 5.2 with $\mathcal{A}$ as the set of all partial computable structures in some language and E the computable isomorphism relation, which shows that the class of all partial computable structures in any fixed language embeds into $\cong^c_\omega$. As a result, if we consider *tc*-embeddings only in the context of computable isomorphism, *every* class embeds into $\cong^c_\omega$. This gives us the following corollary to Theorem 5.2.

Corollary 5.4. $\left(\cong^c_{\omega\times 2}\right) \leq_{tc} (\cong^c_\omega)$.

Proof. The relation $\cong^c_{\omega\times 2}$ is Σ_3 and therefore satisfies the conditions of Theorem 5.2. □

Theorem 5.2 shows that there is little hope for an interesting structure of $\leq_{tc}$ in the context of computable isomorphism, since even a structure as simple as ω as an ordering is on top. Slight modifications to the proof can be used to show that many other very simple structures are also on top, such as the graph with infinitely many singletons and infinitely many connected pairs of vertices of degree 1.

Corollary 5.5. *A relation on a class of uniformly computable structures embeds into $\cong^c_\omega$ if and only if it is Σ_3.*

Proof. Theorem 5.2 tells us that if such a relation is Σ_3, it embeds into $\cong^c_\omega$. On the other hand, assume we have a relation E on a uniformly computable class $\mathcal{A} = \{A_0, A_1, \ldots\}$ of structures and a *tc*-embedding f from E to $\cong^c_\omega$. We can find a Σ_3 predicate $P(x, y)$ corresponding to E by first mapping A_x and A_y to indices for $f(A_x)$ and $f(A_y)$ (which we can compute) and then using the fact that $\cong^c_\omega$ is Σ_3. □

We now turn to proving Theorem 5.2. The proof proceeds in two main steps. In the first step, handled by Lemma 5.8, we construct a *tc*-embedding on indices. The formal definition will be given below, but a *tc*-embedding on indices is essentially a *tc*-embedding that instead of mapping diagrams of structures, maps indices of computable structures. This *tc*-embedding on indices embeds the class of all partial

computable structures in some language into the class of computable copies of ω. While previous results about *tc*-embeddings generally built embeddings by guessing at invariants of individual structures, in this proof we guess instead at the relations between the structures.

The second main step, which will be handled after Lemma 5.8, is converting the embedding on indices into a *tc*-embedding. We will do so by another tree construction which takes the diagram of a structure and tries to "match it up" with its index while simultaneously building a copy of ω based on these guesses and on the embedding on indices constructed in the first step.

Definition 5.6. Let E_A and E_B be equivalence relations on ω. A *tc-embedding on indices from* E_A *to* E_B is a computable function $f : \omega \to \omega$ such that

$$e_1 E_A e_2 \iff f(e_1) E_B f(e_2).$$

When applied to indices of structures, this definition is very similar to that of a *tc*-embedding, except that instead of mapping diagrams of structures we are mapping their indices.

Remark 5.7. Given a *tc*-embedding between classes $\mathcal{A}$ and $\mathcal{B}$, we can construct a *tc*-embedding on indices between the indices for computable structures in $\mathcal{A}$ and those in $\mathcal{B}$, but the converse is not necessarily true even when $\mathcal{A}$ and $\mathcal{B}$ contain only computable structures.

We will begin by constructing a *tc*-embedding on indices. In fact, we will prove the following lemma, which tells us that any Σ_3 equivalence relation embeds into the computable isomorphism relation on ω:

Lemma 5.8. *Let R be a Σ_3 equivalence relation on ω and E be the computable isomorphism relation on indices of computable copies of ω. Then $R \leq_{tc} E$.*

Proof. We will construct a uniform sequence $\langle B_i : i \in \omega \rangle$ of computable presentations of ω, with orderings $<_{B_i}$, satisfying the following requirements (where e, i, and j range over ω, with the condition that $i < j$):

$R_{i,j,e}$: If $\neg iRj$, then B_i and B_j are not computably isomorphic by φ_e, and
$S_{i,j}$: If iRj is true, then B_i and B_j are computably isomorphic.

Together, these are stating formally what it means to be a *tc*-embedding on indices. Unfortunately, R is not computable, and so we will need to use the guessing strategy of Remark 2.4 in the construction. Since iRj is Σ_3, it can be written in the form $\exists x \forall y \exists z\, Q(i, j, x, y, z)$, where Q is computable. A *witness* for iRj is an e such that the Π_2 statement $\forall y \exists z\, Q(i, j, e, y, z)$ is true; it is these statements (or their negations, which are Σ_2) that we will guess at. We will organize our guesses on a tree, which we will call the *tree of outcomes*. Edges on the tree represent guesses about whether or not a certain value e is a witness for iRj; paths through the tree represent a set of guesses about all of R. Associated with each node of the tree is one of the requirements $R_{i,j,e}$ or $S_{i,j}$.

There are two kinds of outcomes (again restricting to $i < j$):

(1) outcomes of the form "e is not a witness for iRj" and
(2) outcomes of the form "iRj".

Since we may never know for a given i, j whether iRj, we cannot simply run the guessing strategy for *all* of the outcomes. To see why, imagine that at some

stage we guess that iRj but also guess that it is not the case that jRi; we would try to act to make B_i and B_j computably isomorphic while simultaneously trying to make them non-isomorphic. To get around this issue, we will construct our tree of outcomes so that every path through it specifies a set of outcomes with no contradictions or redundancy.

Formally, given a set of outcomes Γ of the two types above, an outcome (which does not need to be—and which generally will not be—contained in Γ) will be said to be *redundant* with Γ if at least one of the following holds:

- It is of the form "iRj" but was already implied by elements of Γ based on the fact that R is an equivalence relation; that is, there is a sequence $i = i_0, i_1, \ldots, i_n = j$ so that for each $0 \leq k < n$, Γ contains an outcome of the form $i_k R i_{k+1}$.
- It is of the form "e is not a witness for iRj" but Γ already contains that outcome or contains an outcome of the form "kRi" or an outcome of the form kRj (in other words, either i or j is not minimal in its class according to the outcomes in Γ).

This corresponds exactly to our intuitive notion of what it means for the outcome to already be implied by the outcomes contained in Γ, given that R is an equivalence relation.

We say an outcome of the form "e is not a witness for iRj" is *inconsistent* with Γ if the outcome "iRj" is redundant with Γ. We say that an outcome is *extraneous* for Γ if it is redundant or inconsistent with Γ. Giving some ordering of order type ω to these outcomes, we can finally define the tree of outcomes: the right edge of some node will be the least outcome of the form "e is not a witness for iRj" not above it in the tree, and the left edge from that node is of the form "iRj" with the same i and j. Note that any path through the tree gives just enough information to completely determine R, specifying isomorphisms between each element and the minimum element of its class and non-isomorphisms between pairs of elements that are minimal in their classes.

Any node on the tree of outcomes directly below an edge in the tree associated with the outcome "e is not a witness for iRj" is associated with the requirement $R_{e,i,j}$, while any node directly below an outcome of the form "iRj" is associated with $S_{i,j}$. This corresponds to the fact that when we guess that e is not a witness for iRj we will want to act to ensure that φ_e is not a computable isomorphism between B_i and B_j. Similarly, when we guess that iRj we want to take steps toward making B_i and B_j computably isomorphic.

At a given stage s in the construction, let f_s be the binary string of length s, thought of as the path through the tree of outcomes corresponding to the guesses at stage s. Formally, we define $f_s(0)$ to be the guess at stage s about the outcome at the root node of the tree (0 if the "iRj" outcome seems true, and 1 otherwise). To define $f_s(x+1)$, let n be the node on the tree corresponding to $f_s \upharpoonright x$, and let s' be the least stage not yet used in guessing at the outcomes for node n. We let $f_s(x+1)$ correspond to the guess for the node n at this stage s'. (We do this "slowing down" of the guessing strategies rather than using each node's guess at stage s to ensure that $\liminf_s f_s$ is equal to the leftmost path visited infinitely often.) The *accessible nodes* are those lying on f_s; they correspond to information that "seems correct" at stage s. Every node on the tree is associated with some requirement, but only accessible nodes will ever act to satisfy their requirements.

We will start by defining the *restraint functions* $r_k(s,i)$ where k is a node on the tree (associated with a requirement of the form $R_{e,i,j}$ or $S_{i,j}$), s corresponds to the stage, and i is a structure. The way to interpret "$r_k(s,i) = n$" is "no requirement of lower priority than the kth, after stage s, may insert a new element into B_i before the nth position". We let

$$r_k(0,i) = 0$$

for all k; that is, at the beginning of the construction there is no restraint. If a requirement is prevented from being able to act by restraint imposed by a higher-priority argument (that is, a requirement above or to the left on the tree) or a higher-priority argument violates its restraint, we say that requirement is "injured". We will also define "commitment functions" $c_k(s,a)$ for requirements k of the form $S_{i,j}$, corresponding to computable partial isomorphisms between B_i and B_j that are filled out during the construction. We let $c_k(0,a)$ be undefined for all a, since at the beginning of the construction no progress has been made toward making any pair of structures computably isomorphic. Once an element is committed it remains committed forever, and so we will always let $c_k(s+1,a) = c_k(s,a)$ if the latter is defined. For any node k not accessible at a stage s, we will always define $r_k(s,i) = r_k(s-1,i)$ for all i.

For each accessible node associated with $k = S_{i,j}$, we must work to make B_i and B_j isomorphic. The idea will be to "commit" elements of B_j to elements of B_i, so that if B_i ever changes, B_j will change in a corresponding way. This way, as soon as an element is committed (and as long as the requirement is never injured), we know where to map that element to build an isomorphism. B_i itself is completely unaffected. To make all of this precise, let a be an element of B_j such that:

(1) a is uncommitted; that is, $c_k(s,a)$ is undefined; and
(2) a is the least uncommitted element; that is, for all $b \in B_j$ such that $b < a$ in the ordering of B_j, $c_k(s,b)$ is defined.

If no such element exists, add a new element a to the end of B_j (that is, with $a > b$ for all b that are in B_j at this stage); this new element must satisfy both properties. Now assume a is in the nth element of B_j; that is, a has $n-1$ predecessors in the ordering $<_{B_j}$. Set the restraint function at this stage to be

$$r_k(s,j) = \max\{r_k(s-1,j), n\}.$$

Finally, when at some stage $s' \geq s$ we have that B_i has at least n elements, commit a by letting $c_k(s',a)$ be the nth element of B_i.

At every stage $s' > s$, every node k associated with a requirement $S_{i,j}$ active at stage s', and every a so that $c_k(s',a)$ is defined, if the position of a in B_j is less than the position of $c_k(s',a)$ in B_i, then insert a new element b in B_j such that $b < a$ but $b > c$ for all $c < a$, and let $r_k(s',j) = r_k(s'-1,j) + 1$ unless prevented from doing so by restraint imposed by a higher-priority requirement. In this way, we are keeping a in the same position as the element it was committed to whenever possible.

For each accessible node associated with $R_{e,i,j}$, we will act to make B_i and B_j non-isomorphic by φ_e, as follows: add a pair of elements a, b to B_i, where $a <_{B_i} b$ and both are $<_{B_i}$-greater than any existing element of B_i. If b is the nth element

of B_i (according to $<_{B_i}$), let

$$r_k(s,i) = n,$$

preventing any lower-priority requirements from adding elements between (or before) a and b. During the rest of the construction, we wait until $\varphi_e(a)$ and $\varphi_e(b)$ both converge, if ever. If at some future stage s' we have $\varphi_e(a) = a'$ and $\varphi_e(b) = b'$, we act as follows:

(1) If $a' >_{B_j} b'$ or if there is c' such that $a' <_{B_j} c' <_{B_j} b'$, do nothing: we have already ensured that φ_e is not an isomorphism between B_i and B_j, because it fails on the elements a and b.

(2) If $a' <_{B_j} b'$ are adjacent in B_j and b' is in the nth position in B_j, add a new element c to B_i such that $a <_{B_i} c <_{B_i} b$. Define

$$r_k(s',i) = r_k(s'-1,i) + 1$$

and

$$r_k(s',j) = \max\{r_k(s'-1,j), n\}.$$

Once again we have ensured that φ_e fails to be an isomorphism on a and b, because it maps non-adjacent elements to adjacent ones.

One obstacle may be that we are prevented from inserting c by restraint of higher-priority requirements. If this happens, we start anew with the requirement, adding a new a and b.

This completes the construction. We must now verify that the construction satisfies all of the requirements.

Define the true path f to be the leftmost path through the tree that is visited infinitely often during the construction, so that f_s only finitely often goes to the left of any node on f but infinitely often passes through any particular node on it. Because of the "slowing down" in the definition of f_s, $f(n) = \liminf_s f_s(n)$, which, by our guessing strategy, corresponds to the true outcome at the node n.

Let k be a node on the true path, associated with the requirement $S_{i,j}$. We need to show that B_i and B_j end up being computably isomorphic. Because k is on the true path, $S_{i,j}$ will act infinitely many times, and only finitely many times will nodes to the left of it act. Let s be a stage after which no nodes to the left of k will ever act again. While nodes to the left of k on the tree may apply restraint which interferes with $S_{i,j}$, the construction ensures that no node above k will, and so after stage s, $S_{i,j}$ will no longer be restrained. Since k is visited infinitely often, and each time we visit k we commit the least uncommitted element, every element will eventually be committed. It follows that

$$g(a) = \bigcup_s c_k(s,a),$$

a union of compatible partial functions, is defined for all a, is itself computable, and agrees with the isomorphism between B_i and B_j on all but the finitely many a that were restrained before stage s. It follows that the isomorphism between B_i and B_j is computable.

On the other hand, if the true path contains no node associated with $S_{i,j}$ and i and j are both minimal in their equivalence classes, then by the construction of the tree of outcomes the true path contains a node k associated with $R_{i,j,e}$ for all e. Once again, for each e, there is a stage s after which no requirements to the left of k will ever act again. No node above k of the form $S_{a,b}$ can add restraint that

interferes with it—such a node would need to have a or b equal to i or j, which are excluded by the way we built the tree. Furthermore, nodes associated with $R_{a,b,e}$ above k will each act only finitely often, and so eventually, by some stage $s > s'$, k will no longer ever be injured by any of them either. Therefore $R_{i,j,e}$ will never be injured after the stage s', and so φ_e will not be an isomorphism between B_i and B_j. Since every index for a computable isomorphism is ruled out, B_i and B_j will not be computably isomorphic.

Finally, if iRj, then by the construction of the tree there are $i = i_0, i_1, \ldots, i_n = j$ such that for all $0 \leq k < n$ either $S_{i_k,i_{k+1}}$ or S_{i_{k+1},i_k} lie on the true path. Therefore B_{i_k} and $B_{i_{k+1}}$ will be computably isomorphic for all k, and so B_i will be computably isomorphic to B_k. A similar argument shows that if $\neg iRj$, then B_i and B_j will not be computably isomorphic.

It remains to show that the minimal structure in each equivalence class is (classically) isomorphic to ω. This is sufficient because the argument above shows that each structure is computably isomorphic to one of these structures.

Let x be some element introduced into B_i for any reason. This element must have been inserted in attempting to satisfy a requirement $S_{a,b}$ or a requirement $R_{e,a,b}$. We will show that only finitely many elements are ever inserted before it.

Elements added in satisfying a requirement of the form $S_{a,b}$ are only added to B_b, so the only requirements we need to worry about when considering B_i are those of the form $S_{j,i}$ with $j < i$, which are of higher priority than the requirement which inserted x (because of the restraint). Each $S_{j,i}$ only adds elements when it is active, but since we are assuming that B_i is minimal in its class, each such requirement is active only finitely often. In other words, there is a stage after which all requirements $S_{j,i}$ of higher priority than the requirement which inserted x are ever active. At no later stage will any requirement of the form $S_{j,i}$ insert an element before x.

Elements added in satisfying a requirement of the form $R_{e,a,b}$ are only added to B_a, and so we only need to worry about requirements of the form $R_{e,i,j}$ with $i < j$. Such requirements add elements in two circumstances: they may add a pair of elements to the end of B_i, but such elements will always be added after x, or they may add an element between a pair of existing elements, but the only requirements able to do that are ones that had already been active at the stage at which x was inserted, and each one can insert at most one new element below x. Therefore, the requirements $R_{e,a,b}$ can insert only finitely may elements below x.

Because each element x only ever has finitely many elements inserted before it, it follows that $B_i \cong \omega$. □

The final step in proving Theorem 5.2 is turning the *tc*-embedding on indices into a *tc*-embedding.

Proof of Theorem 5.2. Let $f : \omega \to \omega$ be the *tc*-embedding on indices given by Lemma 5.8. We will build a "true" *tc*-embedding from f.

To build a Turing computable embedding from the above construction, assume we are given the diagram for some structure A, and let A_i be a numbering of the computable structures in the class we are embedding. The idea is to "match up" A with some computably isomorphic A_i using the same guessing strategy as before. Let B_i be the structures corresponding to A_i in the construction above.

We will build a copy B of ω based on these guesses. In this construction, we will only act to satisfy the requirements making B isomorphic to some B_i. Specifically, B will work to become isomorphic to some B_i depending on our guesses about the isomorphism type of A.

Specifically, for each pair (e, i), we can use the usual Π_2 guessing strategy to guess whether φ_e is an isomorphism between A and A_i. If it is, we will infinitely often guess that it is; if not, we will eventually always guess that it is not.

All requirements have the following form:

$$R_{e,i}: \text{ if } A \simeq A_i \text{ by } \varphi_e, \text{ then } B \cong^c B_i.$$

If we choose an arbitrary ordering of the pairs $\langle e, i\rangle$, then $R_{e,i}$ has higher priority than $R_{e',i'}$ if $\langle e, i\rangle < \langle e', i'\rangle$.

Again we will have restraint functions $r(s, e, i)$, with $r(0, e, i) = 0$. We no longer need a tree of strategies. Instead, at some stage $s > 0$ of the construction, there will be a least $\langle e, i\rangle$ such that we are currently guessing that $A \simeq A_i$ by φ_e. We will then act on $R_{e,i}$ as follows:

(1) Let $r(s, e, i) = n$, where n is the number of elements in B at this stage; that is, restrain all of B (so far) against changes by lower-priority requirements.
(2) At every future stage s', if B_i gains a new element in the mth position, where $m < r(s', e, i)$, add a new element in the corresponding position of B and define
$$r(s', e, i) = r(s' - 1, e, i) + 1,$$
unless we are prevented from doing so by restraint imposed by higher-priority requirements; that is, if there are $s'' < s'$ and $\langle e', i'\rangle < \langle e, i\rangle$ such that $r(s'', e', i') \geq m$. If we are prevented, let
$$r(s', e, i) = r(s' - 1, e, i).$$

Intuitively, we are "committing" longer and longer initial segments of B to mirror exactly what A_i does every time we guess that A and A_i are computably isomorphic.

If $\langle e, i\rangle$ is minimal such that $A \simeq A_i$ by φ_e, there is some stage s after which no higher-priority requirement ever acts and after which $R_{e,i}$ acts infinitely many times. We may therefore construct a computable isomorphism between B and B_i as follows, using i (so we can recover A) and e, on input x:

(1) Wait until stage s in this construction.
(2) Wait until a stage when the restraint imposed by $R_{e,i}$ includes x.
(3) Map x to the corresponding element of B_i at that stage.

This shows that B is computably isomorphic to some B_i (and hence classically isomorphic to ω) as long as each x is eventually restrained by $R_{e,i}$. Since $R_{e,i}$ acts infinitely many times, there will be some stage after x has entered B, and after stage s (so higher-priority requirements have stopped acting), in which $R_{e,i}$ acts. But each time $R_{e,i}$ acts it restrains all elements in B by that stage, and so x will eventually be restrained. □

6. Conclusion

Computable isomorphism relations seem at first like a promising setting in which to apply the theory of *tc*-embeddings; after all, computable isomorphism is an effectivization of the concept of an isomorphism, just as *tc*-embeddings are an effectivization of Borel embeddings. But Theorem 5.2 shows that *tc*-embeddings are

not very useful in comparing them—even simple structures like ω are complete. On the other hand, the results of this paper start to tell us about how computable isomorphism fits in with *other* relations. Furthermore, Theorem 5.2 shows that in some cases, even when we have no invariants for structures, we might still be able to characterize which classes embed into them based on the relation on pairs. The same technique may work with other equivalence relations on classes of computable structures, even when (as in this case) tools such as the Pull-back Theorem cannot help.

A few questions are raised by these results. Theorem 5.2 provides a classification of exactly what structures embed into computable copies of $(\omega, <)$ in terms of Σ_3 predicates. This result means that there is little hope for much interesting structure for tc-embeddings in the setting of computable isomorphism. But can we make this statement more precise and classify what structures are on top for tc-embeddings in the context of computable isomorphism?

More generally, even though restricting tc-embeddings to only computable isomorphism appears to be a dead end, how does the computable isomorphism relation fit in with other equivalence relations? Do any computable isomorphism relations embed into (or have embeddings from) other equivalence relations, perhaps even the isomorphism relation on some class of structures?

We might also ask what happens in the case of relative computable isomorphism, where instead of demanding that an isomorphism be computable, we only require that it be computable in the diagrams of the two isomorphic structures. Though it is not an equivalence relation, we can still ask for a tc-embedding that respects it. Once again the Pull-back Theorem does not help (since again we have a finer relation than isomorphism, which is in addition not an equivalence relation). The construction in this paper (which relies very heavily on the fact that we are dealing only with computable structures) also fails.

Acknowledgements

The author would like to thank Denis Hirschfeldt and Robert Soare for their guidance, as well as the referee for giving many helpful suggestions.

References

[1] U. Kalvert, D. Kammins, Dz. F. Naĭt, and S. Miller, *Comparison of classes of finite structures*, Algebra Logika **43** (2004), no. 6, 666–701, 759 (Russian, with Russian summary), DOI 10.1023/B:ALLO.0000048827.30718.2c; English transl., Algebra Logic **43** (2004), no. 6, 374–392. MR2135387 (2006e:03049)

[2] Riccardo Camerlo, *The relation of recursive isomorphism for countable structures*, J. Symbolic Logic **67** (2002), no. 2, 879–895, DOI 10.2178/jsl/1190150114. MR1905171 (2003b:03064)

[3] Ekaterina B. Fokina and Sy-David Friedman, *Equivalence relations on classes of computable structures*, Mathematical theory and computational practice, Lecture Notes in Comput. Sci., vol. 5635, Springer, Berlin, 2009, pp. 198–207, DOI 10.1007/978-3-642-03073-4_21. MR2545894 (2011h:03086)

[4] Ekaterina B. Fokina, Sy-David Friedman, and Asger Törnquist, *The effective theory of Borel equivalence relations*, Ann. Pure Appl. Logic **161** (2010), no. 7, 837–850, DOI 10.1016/j.apal.2009.10.002. MR2601014 (2011m:03089)

[5] Harvey Friedman and Lee Stanley, *A Borel reducibility theory for classes of countable structures*, J. Symbolic Logic **54** (1989), no. 3, 894–914, DOI 10.2307/2274750. MR1011177 (91f:03062)

[6] Greg Hjorth, *Borel equivalence relations*, Handbook of set theory. Vols. 1, 2, 3, Springer, Dordrecht, 2010, pp. 297–332 (Vol. 1), DOI 10.1007/978-1-4020-5764-9_5. MR2768683

[7] Vladimir Kanovei, *Borel equivalence relations. Structure and classification*, University Lecture Series, vol. 44, American Mathematical Society, Providence, RI, 2008. MR2441635 (2009f:03060)

[8] Julia F. Knight, Sara Miller, and M. Vanden Boom, *Turing computable embeddings*, J. Symbolic Logic **72** (2007), no. 3, 901–918, DOI 10.2178/jsl/1191333847. MR2354906 (2008h:03031)

[9] Sara B. Quinn, *Algorithmic complexity of algebraic structures*, Thesis (Ph.D.)–University of Notre Dame, 2008, ProQuest LLC, Ann Arbor, MI. MR2736772

[10] Robert I. Soare, *Recursively enumerable sets and degrees. A study of computable functions and computably generated sets*, Perspectives in Mathematical Logic, Springer-Verlag, Berlin, 1987. MR882921 (88m:03003)

[11] Robert I. Soare, *Computability theory with applications, the art of classical computability*, Springer-Verlag, Heidelberg, 2012, to appear.

Department of Mathematics, University of Chicago, 5734 S. University Avenue, Chicago, Illinois 60637

E-mail address: wrightm@gmail.com