Computable Structures and Isomorphisms

Jonny Stephenson

Valparaiso University

3-29-2018

Computability

A computable function is one given by an **algorithm**.

An algorithm is a finite list of unambiguous instructions which could be carried out by a computer without access to external data.

Think of your favourite programming language...

Usually we work with inputs/outputs in $\mathbb{N}$, but we could also work with pairs, strings, etc.

Computable sets are those with computable characteristic functions.

The Halting Problem

Notice that our description of algorithms doesn't require that they actually succeed.

In general, working out whether an algorithm will halt when provided with a particular input is a hard question.

Formally: Fix a list $\{\varphi_e\}_e$ of all algorithms.

Define $\emptyset' = \{(e, n) : \varphi_e(n) \text{ halts}\}$.

The set $\emptyset'$ is not computable.

Relative computability

We want to have a notion of exactly how noncomputable a set is.

To do so, we now consider the idea of an algorithm which is allowed to access an external noncomputable **oracle**.

We say that A is computable from B, or that B can compute A, if there is an algorithm which, when allowed access to B, is able to compute A.

Then we write $A \leq_T B$, and say A is (Turing) reducible to B.

We say A and B have the same Turing degree if $A \leq_T B$ and $B \leq_T A$.

EG suppose $A \subseteq \mathbb{N}$ and $B = \{n^2 \mid n \in A\}$. Then $A \equiv_T B$.

Sets with the same Turing degree can compute the same things.

An interpretation of the Halting Problem

One way to interpret $\emptyset'$ is in terms of searches.

For instance, consider the following algorithm:

On input n, search for and list out n pairs of twin primes.

This is a computable procedure, but for large values of n we *don't know* whether there are n pairs of twin primes or not, hence whether or not the search succeeds.

Using $\emptyset'$ we can determine whether any particular search will ever halt.

To tell whether there are *infinitely* many twin primes needs a stronger oracle than $\emptyset'$. The appropriate set is called $\emptyset''$.

Computable structures

Computable structure theory focuses on **computable** structures.

Definition

A computable structure $\mathcal{A}$ consists of a computable domain, and computable relations and functions.

Most "everyday structures" of mathematics are computable.

EG: the linear orders $\mathbb{Q}$ and $\mathbb{N}$ (or these sets with their algebraic structures).

There are structures without any computable copy.

Computable isomorphisms

We study computable structures up to *computable* isomorphism.

We are interested in the Turing degree of isomorphisms between different computable copies of the same structure.

Computable categoricity

Definition

We say that a computable structure is computably categorical if there is a computable isomorphism between any two computable copies.

Example

Any two computable copies of the order $\mathbb{Q}$ are computably isomorphic.

Noncomputable isomorphisms

Many standard structures have pairs of computable copies which are not computably isomorphic.

We make a study of $\mathbb{N}$ as a linear order:

Example

Isomorphisms between computable copies of the structure are always $\emptyset'$-computable.

There are computable copies with isomorphism of Turing degree $\emptyset'$ between them.

So $\emptyset'$ provides exactly the information needed to compute isomorphisms between computable copies.

On closer inspection

We used the usual copy $\mathcal{N}$ and built a strange computable copy $\mathcal{A}$, with isomorphism $f : \mathcal{A} \to \mathcal{N}$.

There's a computable subset U of $\mathcal{A}$ for which $f(U)$ can compute $\emptyset'$.

But $\emptyset'$ can compute f, and therefore from $f(U)$ we can compute f.

Degrees of Categoricity

We formalize the question "how hard is it to find an isomorphism" as a definition:

Definition

The structure $\mathcal{A}$ is $\mathbf{c}$*-computably categorical if* $\mathbf{c}$ *can compute an isomorphism between any two computable copies of* $\mathcal{A}$*.*

The degree of categoricity of the structure $\mathcal{A}$ *is* $\mathbf{d}$ *if* $\mathcal{A}$ *is* $\mathbf{c}$*-computably categorical precisely when* $\mathbf{d} \leq_T \mathbf{c}$*.*

Essentially, degree of categoricity is exactly the right amount of information to always be able to find isomorphisms between computable copies.

What do we learn from degrees of categoricity?

There are often strong links between degrees of categoricity and structural features.
For instance: to build an isomorphism between two copies of the natural numbers, we need to know the least element and the successor relation.

Both of these can be expressed as single-quantifier facts:
"Is there anything smaller?"
"Is there anything in between?"

These correspond to computable searches, so $\emptyset'$ can answer the questions.

Less natural difficulties?

There are other structures with degree of categoricity $\emptyset'$, but where the difficulty in finding an isomorphism isn't associated with searches of this kind.

Instead, we can build a computable structure where building an isomorphism requires specific information about $\emptyset'$ itself, rather than answering one-quantifier questions about the structure.

A general phenomenon?

When finding the degree of categoricity of the natural numbers, we coded the degree of the isomorphism f into $f(U)$ for a computable subset U of our structure.

This is quite a common theme in constructions of computable structures.

We (Csima, Deveau, S.) asked:

Question

Is it generally true that if $f : \mathcal{A} \to \mathcal{B}$ is an isomorphism of computable structures, then $f \equiv_T f(U)$ for some computable subset U of $\mathcal{A}$?
If not, when is it true?

We studied some examples for motivation.

Theorem

There are computable copies $\mathcal{A}$ and $\mathcal{B}$ of the natural numbers with isomorphism $f : \mathcal{A} \to \mathcal{B}$ and such that:
There is no computable subset U of $\mathcal{A}$ with $f(U) \equiv_T f$.
There is no computable subset U of $\mathcal{B}$ with $f^{-1}(U) \equiv_T f$.
Furthermore, $f \equiv_T \emptyset'$.

The previous result *requires* the use of "unusual" copies of the natural numbers.

Here, we refer to $\mathcal{N}$ as the usual computable copy of the natural numbers.

Theorem

Let $\mathcal{A}$ be a computable copy of the natural numbers. Let $f : \mathcal{A} \to \mathcal{N}$ be the isomorphism between them.
Then there is a computable subset U of $\mathcal{A}$ with $f(U) \equiv_T f$.

Curiously, there is a fundamental asymmetry here: if we try to reverse the direction of the isomorphism, the result no longer is true.

Thank you!