An Introduction to Computability Enumerable Equivalence Relations

Caleb Ziegler

Abstract

This presentation will present an overview of the theory of computably enumerable equivalence relations (ceers). First, we define recursive and partial recursive functions and introduce the enumeration. Then we discuss computable and computably enumerable sets. Once these are defined we can define ceers and introduce the partial order on these sets. Once these terms are defined, we will discuss the theory of ceers, such as constructing explicit ceers that we use to understand the order and attempting to understand various jumps.

1 Introduction to Computability

We have several equivalent notions of computability. The intuitive idea of a computable function is a function that is can be calculated by a deterministic mechanical process. The canonical definitions of computable functions are those computable by recursive functions and Turing machines, but there are many other processes that provably compute the same functions. Since we cannot prove statements regarding actual mechanical processes, we have the Church-Turing Thesis, which states that the functions that are computable correspond to the functions that can be computed by Turing machines or some system equivalent to Turing machines. Using this thesis, we can abstract away from the particulars of the machinery we use to compute functions and just use an arbitrary appropriate enumeration.

Definition A partial recursive function is defined as a function that can be computed by a Turing machine. A partial recursive function f does not necessarily take a value for every x. In the case where a value is undefined, we say that the computation does not halt and denote it $f(x)\uparrow$. Similarly, if f halts on x, we denote this $f(x)\downarrow$. A partial recursive function that halts for every value of x is called total. We denote an enumeration of the partial recursive functions φ_x. An example of such an enumeration is listing Turing machines by a numbering of the instructions. This type of simple listing enables to use the instructions indexed by x in defining functions. We denote these instructions P_x.

Note that this shows that there are countably many partial recursive functions, since there are only countably many possible finite sets of instructions. Also, we have that there are countably many indices for a given function, since we can create a new index for a given set of instructions by adding dummy instructions to the end.
From such a canonical definition of an enumeration, we get the theorems that enable much of our analysis: the S^m_n Theorem and the Fixed Point Theorem.

Theorem 1 *S^m_n Theorem: For any partial recursive function $\psi(x_1,..,x_m,y_1,...,y_n)$, there is a total recursive function $S(x_1,...,x_m)$ such that, $\varphi_{S(x_1,...,x_m)}(y_1,...,y_m) = \psi(x_1,...,x_m,y_1,...,y_n)$ for all values of the x_i and y_j.*

Theorem 2 *Fixed Point Theorem: For any total recursive function $f(x_1,...,x_{n+1})$, there is a computable function n_f, such that for all values of $x_1,...,x_n$, we have that $\varphi_{f(x_1,...,x_n,n_f(x_1,...,x_n))} = \varphi_{n_f(x_1,...,x_n)}$. In particular, for one variable, there exists an index e such that $\varphi_{f(e)} = \varphi_e$.*

Note that a single variable generalises to any number of variables because we can define a recursive coding from the space of finite sequences of elements from $\mathbb{N}$ to $\mathbb{N}$. We denote such a coding of $x_1,...,x_n$ by $< x_1,...,x_n >$.
In this analysis, we did not specify the details of the enumeration used. Such a definition is justified by the Roger's Isomorphism Theorem.

Theorem 3 *Roger's Isomorphism Theorem: Let φ_x be the canonical enumeration. If ψ_x is another enumeration that satisfies the S^m_n theorem and the Fixed Point Theorem, then there exists a recursive isomorphism σ such that $\psi_x = \varphi_{\sigma(x)}$.*

Because of this, we see that we can consider an arbitrary enumeration while still using intuition from Turing machine instructions to define new partial recursive functions.
Next, we characterise sets that are computable.

Definition We call a set R recursive if its characteristic function

$$f(x) = 1, \text{ for } x \in R, \; f(x) = 0 \text{ for } x \notin R,$$

is recursive. A set R is recursively enumerable if it is the domain of a partial recursive function. We can enumerate the recursively enumerable sets by $W_x = domain(\varphi_x)$.

From these definitions, we see immediately that if a set is recursive, then it is recursively enumerable and a set R is recursive if and only if R and $\mathbb{N} \backslash R$ are both recursively enumerable.

Theorem 4 *There exist recursively enumerable, non-recursive sets. In particular, $K = \{x | \varphi \downarrow\}$ is recursively enumerable, but not recursive.*

With this background in place, we can define ceers.

2 Ceers

Definition A ceer is a recursively enumerable set $R = < x, y >$ satisfying:
$\forall < x, x > \in R$,
$< x, y > \in R \implies < y, x > \in R$, and
$< x, y >, < y, z > \in R \implies < x, z > \in R$.
We often denote $< x, y > \in R$ by xRy. We can enumerate all ceers by letting R_x be the ceer generated by $domain(\varphi_x)$.

Definition For ceers R_1 and R_2, we define a reducibility notion, with R_1 reducible to R_2, denoted $R_1 \leq R_2$, if there exists recursive function $f : \mathbb{N} \to \mathbb{N}$ such that for every $x,\ y \in \mathbb{N}$,

$$xR_1y \text{ if and only if } f(x)R_2f(y).$$

If $R_1 \leq R_2$ and $R_2 \leq R_1$, we write $R_1 \equiv R_2$. A ceer E is called universal, if for every ceer R, we have that $R \leq E$.

There are other definitions of reducibility for ceers, but this definition best mimics the reducibility of sets into computability classes. The simplest ceers are the computable ceers.

Definition We define the ceer $id(n)$ for $n \in \mathbb{N}$, by $xid(n)y \iff x \equiv y \ mod\ n$. Define the ceer ω by $x\omega y \iff x = y$.

These are the only recursive ceers up to equivalence and if R is recursive, then $R \leq \omega$.
An example of a non-recursive ceer is $xHy \iff x = y$ or $\varphi_x(x) \downarrow = \varphi_y(y)$. This leads to a jump that can be applied to an arbitrary ceer.

Definition Let R be a ceer. We define the halting jump of R, denoted R' by $xR'y \iff x = y$ or $\varphi_x(x) \downarrow R\varphi_y(y) \downarrow$. We denote n iterations of the halting jump applied to R by $R^{(n)}$.

It is not difficult to see that $R \leq R'$ for any ceer R.
Other properties include:
For ceers R_1 and R_2, we have that $R_1 \leq R_2 \iff R_1' \leq R_2'$.
R universal $\implies$ R' universal.
For any ceer R, either $R \equiv R^{(n)}$ for every n, or $R^{(n)} < R^{(n+1)}$.
One of the unsolved questions of this theory is whether or not every ceer that satisfies $R \equiv R'$ is universal.
We define the function $\kappa(x) = \varphi_x(x)$, so we can define the halting jump as $xR'y \iff x = y$ or $\kappa(x) \downarrow = \kappa(y) \downarrow$. In considering this problem, it becomes natural to look at repeated iterations of this function. One of the problems from my research involved iteration of κ and the sets that can be expressed in that form.