A user-friendly Introduction to (un)Computability and Unprovability via "Church's Thesis" Part III

0.1. Recursively Enumerable Sets

In this section we explore the rationale behind the alternative name "*recursively enumerable*" —r.e.— or "*computably enumerable*" —c.e.— that is used in the literature for *the semi-recursive or semi-computable* sets/predicates.

To avoid cumbersome codings (of n-tuples, by single numbers) *we restrict attention to the one variable case* in this section.

That is, our predicates are subsets of $\mathbb{N}$.

First we define:

0.1.1 Definition. A set $A \subseteq \mathbb{N}$ is called *computably enumerable* (c.e.) or *recursively enumerable* (r.e.) precisely if one of the following cases holds:

- $A = \emptyset$
- $A = \text{ran}(f)$, where $f \in \mathcal{R}$.

□

Thus, the c.e. or r.e. relations are exactly those we can ***algorithmically enumerate*** as **the set of outputs** of a (*total*) *recursive function*:

$$A = \{f(0), f(1), f(2), \ldots, f(x), \ldots\}$$

Hence the use of the term "c.e." replaces the non technical term "algorithmically" (in "algorithmically" enumerable) by the technical term "computably".

Note that we had to hedge and ask that $A \neq \emptyset$ *for any enumeration to take place*, because no recursive function (remember: these are total) can have an empty range.

Next we prove:

0.1.2 Theorem. ("c.e." or "r.e." vs. semi-recursive) *Any non empty* semi-recursive *relation A ($A \subseteq \mathbb{N}$) is the range of some (emphasis:* **total***)* recursive function *of one variable.*

Conversely*, every set A such that $A = \mathrm{ran}(f)$ —where $\lambda x.f(x)$ is recursive— is* semi-recursive *(and, trivially, nonempty).*

Before we prove the theorem, here is an example:

0.1.3 Example. The set $\{0\}$ is c.e. Indeed, $f = \lambda x.0$, our familiar function Z, effects the enumeration *with repetitions (lots of them!)*

$$\begin{array}{lllllll} x & = 0 & 1 & 2 & 3 & 4 & \ldots \\ f(x) & = 0 & 0 & 0 & 0 & 0 & \ldots \end{array}$$

□

Proof. of Theorem 0.1.2.

(I) **We prove the first sentence of the theorem**. So, let $A \neq \emptyset$ be *semi-recursive*.

By the projection theorem (see Notes #7) there is a **recursive** relation $Q(y, x)$ such that

$$x \in A \equiv (\exists y) Q(y, x), \text{ for all } x \qquad (1)$$

Thus, the totality of the x in A are *the 2nd coordinates of ALL pairs (y, x) that satisfy $Q(y, x)$*.

So, to enumerate all $x \in A$ *just enumerate all pairs (y, x)*, and OUTPUT x just in case $(y, x) \in Q$.

We enumerate *all POSSIBLE PAIRS* y, x by

$$(y = (z)_0, \quad x = (z)_1)$$

for each $z = 0, 1, 2, 3, \ldots$.

Recall that $A \neq \emptyset$. So fix an $a \in A$. f below does the enumeration!

$$f(z) = \begin{cases} (z)_1 & \text{if } Q((z)_0, (z)_1) \\ a & \text{othw} \end{cases}$$

The above is a definition by recursive cases hence f is a recursive function, and the values $x = (z)_1$ that it outputs for each $z = 0, 1, 2, 3, \ldots$ *enumerate* A.

(II) **Proof of the second sentence of the theorem**.

So, let $A = \text{ran}(f)$ —where f is recursive.

Thus,

$$x \in A \equiv (\exists y) f(y) = x \tag{1}$$

By Grz-Ops, plus the facts that *$z = x$ is in $\mathcal{R}_*$ and* the assumption $f \in \mathcal{R}$,

the relation $f(y) = x$ is *recursive*.

By (1) we are done by the Projection Theorem.

□

0.1.4 Corollary. *An* $A \subseteq \mathbb{N}$ *is semi-recursive iff it is r.e. (c.e.)*

Proof. For nonempty A this is Theorem 0.1.2. For empty A we note that this is r.e. by Definition 0.1.1 but is also semi-recursive by $\emptyset \in \mathcal{PR}_* \subseteq \mathcal{R}_* \subseteq \mathcal{P}_*$. □

Corollary 0.1.4 allows us to prove some non-semi-recursiveness results by good old-fashioned Cantor diagonalisation.

See below.

0.1.5 Theorem. *The complete index set* $A = \{x : \phi_x \in \mathcal{R}\}$ *is not semi-recursive.*

This sharpens the undecidability result for A that we established in Note #7.

Proof. Since *c.e.* $=$ *semi-recursive*, we will prove instead that A is *not* c.e.

If not, note first that $A \neq \emptyset$ —e.g., $S \in \mathcal{R}$ and thus all ϕ-indices of A are in A.

Thus, theorem 0.1.2 applies and **there is an** $f \in \mathcal{R}$ **that enumerates** A:

$$A = \{f(0), f(1), f(2), f(3), \ldots\}$$

The above says: ALL programs for unary $\mathcal{R}$-functions are $f(i)$'s.

Define now

$$d = \lambda x.1 + \phi_{f(x)}(x) \tag{1}$$

Seeing that $\phi_{f(x)}(x) = U^{(P)}(f(x), x)$ —*you remember* $U^{(P)}$*?*— we obtain $d \in \mathcal{P}$.

But $\phi_{f(x)}$ is total since *all the* $f(x)$ *are* ϕ*-indices of total functions* by the underlined *blue* comment above. By the same comment,

$$d = \phi_{f(i)}, \text{ for some } i \tag{2}$$

Let us compute $d(i)$: $d(i) = 1 + \phi_{f(i)}(i)$ *by (1)*.

Also, $d(i) = \phi_{f(i)}(i)$ *by (2)*,

thus

$$1 + \phi_{f(i)}(i) = \phi_{f(i)}(i)$$

which is a contradiction *since both sides of "=" are defined*. □

One can take as d different functions, for example, either of $d = \lambda x.42 + \phi_{f(x)}(x)$ or $d = \lambda x.1 \dot{-} \phi_{f(x)}(x)$ works. And infinitely many other choices do!

Lecture #17, Nov. 16

0.2. Some closure properties of decidable and semi-decidable relations

We already *know* that

0.2.1 Theorem. *$\mathcal{R}_*$ is closed under all Boolean operations, $\neg, \wedge, \vee, \rightarrow, \equiv$, as well as under $(\exists y)_{<z}$ and $(\forall y)_{<z}$.*

How about closure properties of $\mathcal{P}_*$?

0.2.2 Theorem. *$\mathcal{P}_*$ is closed under $\wedge$ and $\vee$. It is also closed under $(\exists y)$, or, as we say, "under projection".*

Moreover it is closed under $(\exists y)_{<z}$ and $(\forall y)_{<z}$.

It is **not** *closed under negation (complement),* **nor** *under $(\forall y)$.*

Proof.

1. Let $Q(\vec{x}_n)$ be verified by a URM M, and $S(\vec{y}_m)$ be verified by a URM N.

   Here is how to semi-decide $Q(\vec{x}_n) \vee S(\vec{y}_m)$:

   Given input $\vec{x}_n, \vec{y}_m$, we call machine M with input $\vec{x}_n$, and machine N with input $\vec{y}_m$ and let them crank simultaneously (as "co-routines").

   If **either one** halts, then halt everything! This is the case of "yes" (input verified).

2. For $\wedge$ it is almost the same, but our halting criterion is different:

   Here is how to semi-decide $Q(\vec{x}_n) \wedge S(\vec{y}_m)$:

   Given input $\vec{x}_n, \vec{y}_m$, we call machine M with input $\vec{x}_n$, and machine N with input $\vec{y}_m$ and let them crank simultaneously ("co-routines").

   If **both** halt, then halt everything!

By CT, each of the processes in 1. and 2. can be implemented by some URM.

3. **The $(\exists y)$ is very interesting as it relies on the Projection Theorem:**

   Let $Q(y, \vec{x}_n)$ be **semi**-decidable. Then, by Projection Theorem, a **decidable** $P(z, y, \vec{x}_n)$ exists such that

   $$Q(y, \vec{x}_n) \equiv (\exists z)P(z, y, \vec{x}_n) \tag{1}$$

   It follows that

   $$(\exists y)Q(y, \vec{x}_n) \equiv (\exists y)(\exists z)P(z, y, \vec{x}_n) \tag{2}$$

   This does *not* settle the story, as *I cannot readily conclude* that $(\exists y)(\exists z)P(z, y, \vec{x}_n)$ is semi-decidable ▶because the Projection Theorem requires a *single* $(\exists y)$ in front of a decidable predicate!

   Well, instead of saying that there are **two** values z and y that verify (along with $\vec{x}_n$) the predicate $P(z, y, \vec{x}_n)$, *I can say there is a PAIR of values* (z, y).

   In fact I can CODE the pair as $w = \langle z, y \rangle$ *and say* there is ONE value, w:

   $$(\exists w)P(\overbrace{(w)_0}^{z}, \overbrace{(w)_1}^{y}, \vec{x}_n)$$

   and thus I have —by (2) and the above—

   $$(\exists y)Q(y, \vec{x}_n) \equiv (\exists w)P((w)_0, (w)_1, \vec{x}_n) \tag{3}$$

But since $P((w)_0, (w)_1, \vec{x}_n)$ is **recursive** by the decidability of P *and* Grz-Ops, we end up in (3) quantifying the decidable $P((w)_0, (w)_1, \vec{x}_n)$ with just **one** $(\exists w)$. The Projection Theorem now applies!

4. For $(\exists y)_{<z} Q(y, \vec{x})$, where $Q(y, \vec{x})$ is semi-recursive, we first note that

$$(\exists y)_{<z} Q(y, \vec{x}) \equiv (\exists y)\Big(y < z \wedge Q(y, \vec{x})\Big) \qquad (*)$$

By $\mathcal{PR}_* \subseteq \mathcal{R}_* \subseteq \mathcal{P}_*$, $y < z$ is semi-recursive. By closure properties established *SO FAR* in this proof, the rhs of $\equiv$ in $(*)$ is semi-recursive, thus so is the lhs.

5. For $(\forall y)_{<z} Q(y, \vec{x})$, where $Q(y, \vec{x})$ is semi-recursive, we first note that (by Strong Projection) a **decidable** P exists such that

$$Q(y, \vec{x}) \equiv (\exists w) P(w, y, \vec{x})$$

By the above equivalence, we need to prove that

$$(\forall y)_{<z} (\exists w) P(w, y, \vec{x}) \text{ is semi-recursive} \qquad (**)$$

$(**)$ says that

for **each** $y = 0, 1, 2, \ldots, z-1$ there is a w-value w_y —likely dependent on y— so that $P(w_y, y, \vec{x})$ holds

Since all those w_y are finitely many (z many!) there is a value u bigger than **all** of them (for example, take $u = \max(w_0, \ldots, w_{z-1}) + 1$). Thus $(**)$ says (i.e., **is equivalent to**)

$$(\exists u)(\forall y)_{<z} (\exists w)_{<u} P(w, y, \vec{x})$$

The blue part of the above is **decidable** (by closure properties of $\mathcal{R}_*$, since $P \in \mathcal{R}_*$ —you may peek at 0.2.1). We are done by *strong projection*.

6. Why is $\mathcal{P}_*$ *not closed under negation* (complement)?

   Because we know that $K \in \mathcal{P}_*$, but also know that $\overline{K} \notin \mathcal{P}_*$.

7. Why is $\mathcal{P}_*$ not closed under $(\forall y)$?

   Well,
   $$x \in K \equiv (\exists y)Q(y, x) \tag{1}$$

   for some recursive Q (Projection Theorem) and *by the known fact (quoted again above) that $K \in \mathcal{P}_*$*.

   (1) is equivalent to
   $$x \in \overline{K} \equiv \neg(\exists y)Q(y, x)$$
   which in turn is equivalent to
   $$x \in \overline{K} \equiv (\forall y)\neg Q(y, x) \tag{2}$$
   Now, by closure properties of $\mathcal{R}_*$ See 0.2.1), $\neg Q(y, x)$ is recursive, hence also is in $\mathcal{P}_*$ since $\mathcal{R}_* \subseteq \mathcal{P}_*$.

   Therefore, if $\mathcal{P}_*$ were closed under $(\forall y)$, then the above $(\forall y)\neg Q(y, x)$ *would be semi-recursive*.
   But that is $x \in \overline{K}$! □

0.3. Some tricky reductions

This section highlights a more sophisticated reduction scheme that *improves our ability to effect reductions of the type* $\overline{K} \leq A$.

0.3.1 Example. Prove that $A = \{x : \phi_x \text{ is a constant}\}$ is *not semi-recursive*. This is not amenable to the technique of saying "OK, if A is semi-recursive, then it is r.e. Let me show that it is not so by diagonalisation". This worked for $B = \{x : \phi_x \text{ is total}\}$ but *no obvious diagonalisation comes to mind for A.*

Nor can we simplistically say, OK, start by defining

$$g(x, y) = \begin{cases} 0 & \text{if } x \in \overline{K} \\ \uparrow & \text{othw} \end{cases}$$

The problem is that if we plan next to say "by CT g is partial recursive *hence by S-m-n, etc.*", *we shouldn't!*

The underlined part is wrong: $g \notin \mathcal{P}$, *provably*!

▶ For if it *is* computable, then so is $\lambda x.g(x, x)$ by Grz-Ops.

But

$g(x, x) \downarrow$ iff we have the top case, iff $x \in \overline{K}$

In short,

$$x \in \overline{K} \equiv g(x, x) \downarrow$$

which proves that $\overline{K} \in \mathcal{P}_*$ using the verifier for "$g(x, x) \downarrow$". **Contradiction**. □

0.3.2 Example. (0.3.1 continued) Now, "**Plan B**" is to "**approximate**" the top condition $\phi_x(x) \uparrow$ (same as $x \in \overline{K}$).

The idea is that, "**practically**", if the computation $\phi_x(x)$ after a "huge" number of steps y has still not hit **stop**, this situation *approximates* —let me say once more, "practically"— the situation $\phi_x(x) \uparrow$. This fuzzy thinking suggests that we try next

$$f(x, y) = \begin{cases} 0 & \text{if } \phi_x(x) \text{ did not return in } \leq y \text{ steps} \\ \uparrow & \text{othw} \end{cases}$$

If the top condition is true for a given x it means that at step y the URM that we picked to compute $\phi_x(x)$ has not hit **stop** yet.

The "othw" says, of course, that the computation of the call $\phi_x(x)$ —or $U^{(P)}(x, x)$— *did return in y steps or fewer*.

Next step is to invoke an S-m-n theorem application, so we must show that f defined above is computable. Well here is an informal algorithm:

(0) **proc** $f(x, y)$
(1) **Call** $\phi_x(x)$; keep count of computation steps
(2) **Return** 0 if $\phi_x(x)$ did **not return** in $\leq y$ steps
(3) **"Loop"** if $\phi_x(x)$ **returned** in $\leq y$ steps

Of course, the "command" **Loop** means

"transfer to the subprogram" **while** 1=1 **do** { }

By CT, the pseudo algorithm (0)–(3) is implementable as a URM. That is, $f \in \mathcal{P}$.

By S-m-n applied to f there is a recursive k such that

$$\phi_{k(x)}(y) = \begin{cases} 0 & \text{if } \phi_x(x) \text{ did not return in } \leq y \text{ steps} \\ \uparrow & \text{othw} \end{cases} \tag{1}$$

Analysis of (1) in terms of the "key" conditions $\phi_x(x)\uparrow$ and $\phi_x(x)\downarrow$:

(A) Case where $\phi_x(x)\uparrow$.

Then, $\phi_x(x)$ did **not** halt in y steps, for any y!

Thus, by (1), we have $\phi_{k(x)}(y) = 0$, for all y, that is,

$$\phi_x(x)\uparrow \Longrightarrow \phi_{k(x)} = \lambda y.0 \tag{2}$$

(B) Case where $\phi_x(x)\downarrow$. Let $m =$ *smallest* y such that the call $\phi_x(x)$ ended in m steps. Therefore,

- for step counts $y = 0, 1, 2, \ldots, m-1$ the computation of $U^{(P)}(x, x)$ has not yet hit **stop**, so the **top** case of definition (1) holds. We get

  $$\begin{array}{lllll} \text{for } y & = 0, & 1, & \ldots, & m-1 \\ \phi_{k(x)}(y) & = 0, & 0, & \ldots, & 0 \end{array}$$

- for step counts $y = m, m+1, m+2, \ldots$ the computation of $U^{(P)}(x, x)$ has already halted (it hit **stop**), so the **bottom** case of definition (1) holds. We get

  $$\begin{array}{lllll} \text{for } y & = m, & m+1, & m+2, & \ldots \\ \phi_{k(x)}(y) & = \uparrow, & \uparrow, & \uparrow, & \ldots \end{array}$$

  for short:

  $$\phi_x(x)\downarrow \Longrightarrow \phi_{k(x)} = \overbrace{(0, 0, \ldots, 0)}^{\text{length } m} \qquad (3)$$

In

$$\phi_{k(x)} = \overbrace{(0, 0, \ldots, 0)}^{\text{length } m}$$

we depict the function $\phi_{k(x)}$ *as an array of its* m *output values*.

Thus, in Plain English, when $\phi_x(x)\downarrow$, the function $\phi_{k(x)}$ is NOT a constant! Not even total!

Our analysis yielded:

$$\phi_{k(x)} = \begin{cases} \lambda y.0 & \text{if } \phi_x(x)\uparrow \\ \text{not a constant function} & \text{if } \phi_x(x)\downarrow \end{cases} \tag{4}$$

We conclude now as follows for $A = \{x : \phi_x$ is a constant$\}$:

$k(x) \in A$ iff $\phi_{k(x)}$ is a constant iff the top case of (4) applies
iff $\phi_x(x)\uparrow$

That is, $x \in \overline{K} \equiv k(x) \in A$, hence $\overline{K} \leq A$. □

0.3.3 Example. *Prove (again) that $B = \{x : \phi_x \in \mathcal{R}\} = \{x : \phi_x \text{ is total}\}$ is not semi-recursive.*

We piggy back on the previous example and the same f through which we found a $k \in \mathcal{R}$ such that

$$\phi_{k(x)} = \begin{cases} \lambda y.0 & \text{if } \phi_x(x) \uparrow \\ \overbrace{(0, 0, \ldots, 0)}^{\text{length } m} & \text{if } \phi_x(x) \downarrow \end{cases} \tag{5}$$

The above is (4) of the previous example, but we will use different English words to describe the bottom case, which we displayed explicitly in (5).

Note that $\overbrace{(0, 0, \ldots, 0)}^{\text{length } m}$ is a non-recursive (nontotal) function listed as a finite array of outputs. Thus we have

$$\phi_{k(x)} = \begin{cases} \lambda y.0 & \text{if } \phi_x(x) \uparrow \\ \text{nontotal function} & \text{if } \phi_x(x) \downarrow \end{cases} \tag{6}$$

and therefore

$k(x) \in B$ iff $\phi_{k(x)}$ is total iff the top case of (6) applies iff $\phi_x(x) \uparrow$

That is, $x \in \overline{K} \equiv k(x) \in B$, hence $\overline{K} \leq B$. □

0.3.4 Example. We will prove that $D = \{x : \text{ran}(\phi_x) \text{ is infinite}\}$ is *not semi-recursive*.

We (heavily) piggy back on Example 0.3.2 above.

We want to find $j \in \mathcal{R}$ such that

$$\phi_{j(x)} = \begin{cases} \text{inf. range} & \text{if } \phi_x(x) \uparrow \\ \text{finite range} & \text{if } \phi_x(x) \downarrow \end{cases} \qquad (*)$$

OK, define ψ (almost) like f of Example 0.3.2 by

$$\psi(x, y) = \begin{cases} y & \text{if the call } \phi_x(x) \text{ did not return in} \le y \text{ steps} \\ \uparrow & \text{othw} \end{cases}$$

Other than the trivial difference (function name) the important difference is that we force infinite range in the top case by outputting the input y.

The argument that $\psi \in \mathcal{P}$ goes as the one for f in Example 0.3.2. The only difference is that in the algorithm (0)–(3) we change "**Return** 0" to "**Return** y".

The question $\psi \in \mathcal{P}$ settled, by S-m-n there is a $j \in \mathcal{R}$ such that

$$\phi_{j(x)}(y) = \begin{cases} y & \text{if the call } \phi_x(x) \text{ returns in} \le y \text{ steps} \\ \uparrow & \text{othw} \end{cases} \qquad (\dagger)$$

Analysis of (†) in terms of the "key" conditions $\phi_x(x)\uparrow$ and $\phi_x(x)\downarrow$:

(I) Case where $\phi_x(x)\uparrow$.

Then, for all input values y, $\phi_x(x)$ is still not at **stop** after y steps. Thus by (†), we have $\phi_{j(x)}(y) = y$, for all y, that is,

$$\phi_x(x)\uparrow \Longrightarrow \phi_{j(x)} = \lambda y.y \tag{1}$$

(II) Case where $\phi_x(x)\downarrow$. Let m = *smallest* y such that *the call* $\phi_x(x)$ *returned in* m *steps*.

As before we find that for $y = 0, 1, \ldots, m-1$ we have $\phi_{j(x)}(y) = y$, that is,

$$\begin{array}{lllll} \text{for } y & = 0, & 1, & \ldots, & m-1 \\ \phi_{j(x)}(y) & = 0, & 1, & \ldots, & m-1 \end{array}$$

and as before,

$$\begin{array}{lllll} \text{for } y & = m, & m+1, & m+2, & \ldots \\ \phi_{j(x)}(y) & = \uparrow, & \uparrow, & \uparrow, & \ldots \end{array}$$

that is,

$$\phi_x(x)\downarrow \Longrightarrow \phi_{j(x)} = (0, 1, \ldots, m-1) \text{ —finite range} \tag{2}$$

(1) and (2) say that we got (∗) —p.23— above. Thus

$j(x) \in D$ **iff** $\mathrm{ran}(\phi_{j(x)})$ infinite **iff** top case holds, **iff** $\phi_x(x)\uparrow$

Thus $\overline{K} \le D$ via j. □