A user-friendly Introduction to (un)Computability and Unprovability via “Church’s Thesis” Part II

This is Part II of our Uncomputability notes.

We introduce “half-computable” relations $Q(\vec{x})$ here.

These play a central role in Computability.

The term “half-computable” describes them well: For each of these relations there is a URM M that *will halt* *precisely for the inputs* $\vec{a}$ *that make the relation true*:

i.e., $\vec{a} \in Q$ or equivalently $Q(\vec{a})$ is true.

For the inputs that make the relation false — $\vec{b} \notin Q$— M loops forever.

That is, M *verifies* membership but does not *yes/no-decide* it by halting and “printing” the appropriate 0 (yes) or 1 (no).

Can't we tweak M into M' that is a *decider* of such a Q?

No, not in general! For example, the *halting set K* has a verifier

Right? $x \in K \equiv \phi_x(x) \downarrow \equiv U^{(P)}(x,x) \downarrow$.

So *any program M_Y^X* for the partial recursive $\lambda x.U^{(P)}(x,x)$ is a *verifier* for $x \in K$. See also 0.1.2 below.

But we *KNOW* that $x \in K$ has *NO decider*!

Since the "yes" of a verifier M is signaled by halting but the "no" is signaled by looping forever,

the definition below does not require the verifier to print 0 for "yes". *Here "yes" equals "halting".*

0.1. Semi-decidable relations (or sets)

0.1.1 Definition. (Semi-recursive or semi-decidable sets)

A relation $Q(\vec{x}_n)$ is *semi-decidable* or *semi-recursive* —what we called suggestively "*half-computable*" above—

iff

there is a URM, M, which on input $\vec{x}_n$ ***has a* (halting*!*) *computation iff*** $\vec{x}_n \in Q$.

The output of M ***is unimportant!***

A less civilized, but *more mathematically precise* way to say the above is:

A relation $Q(\vec{x}_n)$ is *semi-decidable* or *semi-recursive* iff there is an $f \in \mathcal{P}$ such that

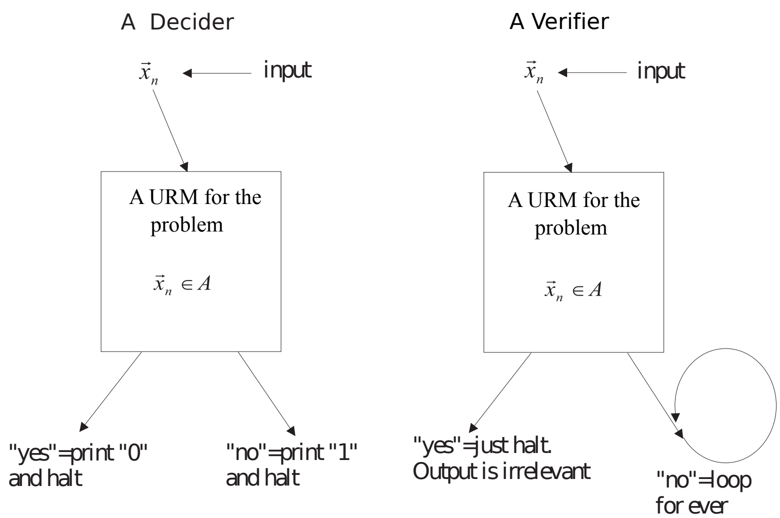
$$Q(\vec{x}_n) \equiv f(\vec{x}_n) \downarrow \tag{1}$$

Clearly, an $f \in \mathcal{P}$ is some $M_{\mathbf{y}}^{\vec{\mathbf{x}}_n}$. Thus, M is a verifier for Q.

The set of *all* semi-decidable relations we will denote by $\mathcal{P}_*$.[†] □

[†]This is not a standard symbol in the literature. Most of the time the set of all semi-recursive relations has *no* symbolic name! We are using this symbol in analogy to $\mathcal{R}_*$—the latter being fairly "standard".

The following figure shows the two modes of handling a query, "$\vec{x}_n \in A$", by a URM.



Here is an important semi-decidable set.

0.1.2 Example. K is semi-decidable. To work within the formal definition (0.1.1) we note that the function $\lambda x.\phi_x(x)$ is in $\mathcal{P}$ via the universal function theorem of Part I: $\lambda x.\phi_x(x) = \lambda x.U^{(P)}(x, x)$ and we know $U^{(P)} \in \mathcal{P}$.

Thus $x \in K \equiv \phi_x(x) \downarrow$ settles it. By Definition 0.1.1 (statement labeled (1)) we are done. □

0.1.3 Example. Any *recursive* relation A *is also semi-recursive*.

That is,

$$\mathcal{R}_* \subseteq \mathcal{P}_*$$

Indeed, intuitively, all we need to do to *convert* a *decider* for $\vec{x}_n \in A$ into a *verifier* is to "intercept" the "print 1"-step and convert it into an "infinite loop",

k : **goto** k

By CT we can certainly do the whole thing via a URM implementation.

A more elegant way (which still invokes CT) is to say, OK: Since $A \in \mathcal{R}_*$, it follows that c_A, its characteristic function, is in $\mathcal{R}$.

Define a new function f as follows:

$$f(\vec{x}_n) = \begin{cases} 0 & \text{if } c_A(\vec{x}_n) = 0 \\ \uparrow & \text{if } c_A(\vec{x}_n) = 1 \end{cases}$$

This is intuitively computable (the "$\uparrow$" is implemented by the same "piece of code" as above).

Hence, by CT, $f \in \mathcal{P}$. But

$$\vec{x}_n \in A \equiv f(\vec{x}_n) \downarrow$$

because of the way f was defined. Definition 0.1.1 rests the case.

One more way to do this: Totally mathematical ("formal", as people say incorrectly[†]) this time!

OK,

$$f(\vec{x}_n) = \text{if } c_A(\vec{x}_n) = 0 \text{ then } 0 \text{ else } \emptyset(\vec{x}_n)$$

That is, using the *sw* function that is in $\mathcal{PR}$ and hence in $\mathcal{P}$, as in

$$\begin{array}{ccccccc} & c_A(\vec{x}_n) & & 0 & & \emptyset(\vec{x}_n) \\ & \downarrow & & \downarrow & & \downarrow \\ f(\vec{x}_n) = \text{if} & z & = 0 \text{ then} & u & \text{else} & w \end{array}$$

$\emptyset$ is, of course, the empty function which by Grz-Ops can have any number of arguments we please! For example, we may take

$$\emptyset = \lambda \vec{x}_n.(\mu y)g(y, \vec{x}_n)$$

where $g = \lambda y \vec{x}_n.SZ(y) = \lambda y \vec{x}_n.1$.

In what follows we will prefer the informal way (proofs by Church's Thesis) of doing things, most of the time. □

An important observation following from the above examples deserves theorem status:

[†]"Formal" refers to syntactic proofs based on axioms. Our "*mathematical*" proofs are mostly *semantic*, depend on meaning, not just syntax. That is how it is in the majority of MATH publications.

0.1.4 Theorem. $\mathcal{R}_* \subset \mathcal{P}_*$

Proof. The $\subseteq$ part of "$\subset$" is Example 0.1.3 above.

The $\neq$ part is due to $K \in \mathcal{P}_*$ (0.1.2) and the fact that the halting problem is unsolvable ($K \notin \mathcal{R}_*$).

So, there are sets in $\mathcal{P}_*$ (e.g., K) that are not in $\mathcal{R}_*$. $\square$

What about $\overline{K}$, that is, the *complement*

$$\overline{K} = \mathbb{N} - K = \{x : \phi_x(x) \uparrow\}$$

of K? *Is it perhaps semi-recursive (*verifiable*)?*

The following general result helps us handle the above question.

0.1.5 Theorem. *A relation $Q(\vec{x}_n)$ is recursive* iff ***both*** *$Q(\vec{x}_n)$ and $\neg Q(\vec{x}_n)$ are semi-recursive.*

Before we proceed with the proof, a remark on notation is in order.

In "set notation" we write the complement of a set, A, of n-tuples as $\overline{A}$. This means, of course, $\mathbb{N}^n - A$, where

$$\mathbb{N}^n = \underbrace{\mathbb{N} \times \cdots \times \mathbb{N}}_{n \text{ copies of } \mathbb{N}}$$

In "relational notation" we write the same thing (complement) as

$$\neg A(\vec{x}_n)$$

Similarly,

"*set notation*": $A \cup B, \quad A \cap B$

"*relational notation*": $A(\vec{x}_n) \vee B(\vec{y}_m), \quad A(\vec{x}_n) \wedge B(\vec{y}_m)$

Back to the proof.

Proof. We want to prove that some URM, N, **decides**

$$\vec{x}_n \in Q$$

We take two *verifiers*, *M for "$\vec{x}_n \in Q$"* and *M' for "$\vec{x}_n \in \overline{Q}$"*,[†] and run them —on input $\vec{x}_n$— as "co-routines" (i.e., we crank them simultaneously).

If M halts, then we stop everything and print "0" (i.e., "yes").

If M' halts, then we stop everything and print "1" (i.e., "no").

CT tells us that we can put the above —if we want to— into a single URM, N. $\square$

[†]We can do that, i.e., M and M' exist, since both Q and $\overline{Q}$ are semi-recursive.

0.1.6 Remark. The above proof handled *only the "if" direction*. For the "only if" this is trivial:

$\mathcal{R}_*$ is *closed under complement (negation)* as we showed way back in a previous Note.

Thus, if Q is in $\mathcal{R}_*$, then so is $\overline{Q}$, by closure under $\neg$. By Theorem 0.1.4, both of Q and $\overline{Q}$ are in $\mathcal{P}_*$. □

0.1.7 Example. $\overline{K} \notin \mathcal{P}_*$.

Now, **this** ($\overline{K}$) is a horrendously unsolvable problem! This problem is so hard it is not even ***semi***-decidable!

Why? Well, if instead it were $\overline{K} \in \mathcal{P}_*$, then combining this with Example 0.1.2 and Theorem 0.1.5 we would get $K \in \mathcal{R}_*$, which we know is not true. □

0.2. Unsolvability via Reducibility

We turn our attention now to a **methodology** towards discovering new undecidable problems, and also new non semi-recursive problems, beyond the ones we learnt about so far, which are just,

1. $x \in K$,

2. $\phi_i = \phi_j$ (equivalence problem)

3. and $x \in \overline{K}$.

In fact, we will learn shortly that $\phi_i = \phi_j$ is worse than undecidable; just like $\overline{K}$ it too is *not even semi-decidable*.

The tool we will use for such discoveries is the concept of *reducibility* of one set to another:

0.2.1 Definition. (Strong reducibility) For any two subsets of $\mathbb{N}$, A and B, we write

$$A \leq_m B^\dagger$$

or more simply

$$A \leq B \tag{1}$$

pronounced *A is strongly reducible to B*, meaning that there is a (total) *recursive* function f such that

$$x \in A \equiv f(x) \in B \tag{2}$$

We say that "*the reduction is effected by f*".

The last sentence has the notation $A \leq^f B$. □

In words, $A \leq_m B$ says that we can *algorithmically* solve the problem $x \in A$ if we know how to solve $z \in B$. The algorithm is:

1. Given x.
2. Given the "subroutine" $z \in B$.
3. Compute $f(x)$.
4. Give the same answer for $x \in A$ (true or false) as *you do for $f(x) \in B$*.

[†]The subscript m stands for "many one", and refers to f. We do not require it to be 1-1, that is; *many* (inputs) to *one* (output) will be fine.

When $A \leq_m B$ holds, then, intuitively,

"A is easier than B to either decide or verify"

since if we know how to *decide* or (only) *verify* membership in B then we can decide or (only) verify membership in A: *"$x \in A$?"*

All we have to do is compute $f(x)$ and ask instead the question "$f(x) \in B$" which we can *decide* or *verify*.

This observation has a very precise counterpart (Theorem 0.2.3 below).

0.2.2 Lemma. *If $Q(y, \vec{x}) \in \mathcal{P}_*$ and $\lambda \vec{z}.f(\vec{z}) \in \mathcal{R}$, then $Q(f(\vec{z}), \vec{x}) \in \mathcal{P}_*$.*

Proof. By Definition 0.1.1 there is a $g \in \mathcal{P}$ such that

$$Q(y, \vec{x}) \equiv g(y, \vec{x}) \downarrow \tag{1}$$

Now, for any $\vec{z}$, $f(\vec{z})$ is some number which if we plug into y in (1) we get an equivalence:

$$Q(f(\vec{z}), \vec{x}) \equiv g(f(\vec{z}), \vec{x}) \downarrow \tag{2}$$

But $\lambda \vec{z}\vec{x}.g(f(\vec{z}), \vec{x}) \in \mathcal{P}$ by Grz-Ops. Thus, (2) and Definition 0.1.1 yield $Q(f(\vec{z}), \vec{x}) \in \mathcal{P}_*$. □

0.2.3 Theorem. *If $A \leq^g B$ in the sense of 0.2.1, then*

(i) if $B \in \mathcal{R}_*$, then also $A \in \mathcal{R}_*$

(ii) if $B \in \mathcal{P}_*$, then also $A \in \mathcal{P}_*$

Proof.

(i) The assumption says that $z \in B$ is in $\mathcal{R}_*$.

So is $g(x) \in B$ by Grz. Ops. (Way back).

But $x \in A \equiv g(x) \in B$, so $x \in A$ is in $\mathcal{R}_*$.

(ii) Let $z \in B$ be in $\mathcal{P}_*$.
By 0.2.2, so is $g(x) \in B$. *But this says $x \in A$.* □

Taking the "contrapositive", we have at once:

0.2.4 Corollary. *If $A \le B$ in the sense of 0.2.1, then*

(i) if $A \notin \mathcal{R}_*$, then also $B \notin \mathcal{R}_*$

(ii) if $A \notin \mathcal{P}_*$, then also $B \notin \mathcal{P}_*$

We can now use K and $\overline{K}$ as a "*yardsticks*" —or *reference* "problems"— and discover *new* undecidable and also *non semi-decidable* problems.

The idea of the corollary is applicable to the so-called "complete index sets".

0.2.5 Definition. (Complete Index Sets) Let $\mathcal{C} \subseteq \mathcal{P}$ and $A = \{x : \phi_x \in \mathcal{C}\}$.

A is thus the set of **ALL** programs (known by their addresses) x that compute any *unary* $f \in \mathcal{C}$:

Indeed, let $\lambda x.f(x) \in \mathcal{C}$. Thus $f = \phi_i$ for some i. Then $i \in A$.

But this is true of **all** ϕ_m that equal f.

We call A a *complete* **index** (programs-) set. □

We embark on several examples, but first note the *FORM of S-m-n Theorem* that we will be using going forward:

0.2.6 Theorem. (S-m-n in practice) *If $\psi \in \mathcal{P}$ has two arguments, then there is a unary $h \in \mathcal{R}$ such that*

$$\psi(x, y) = \phi_{h(x)}(y) \tag{1}$$

for all x, y.

Proof. *Fix an i* such that $\psi(x, y) = \phi_i^{(2)}(x, y)$, for all x, y.

By S-m-n, we have a recursive $\lambda ix.S_1^1(i, x)$ such that

$$\phi_i^{(2)}(x, y) = \phi_{S_1^1(i,x)}(y)$$

for all i, x, y.

But i is fixed.

Thus $\lambda x.S_1^1(i, x)$ is the "h" we want. □

0.2.7 Example. The set $A = \{x : \mathrm{ran}(\phi_x) = \emptyset\}$ is not semi-recursive.

Recall that "range" for $\lambda x.f(x)$, denoted by $\mathrm{ran}(f)$, is defined by

$$\{x : (\exists y) f(y) = x\}$$

We will try to show that

$$\overline{K} \le A \tag{1}$$

If we can do that much, then Corollary 0.2.4, part ii, will do the rest.

Well, define

$$\psi(x, y) = \begin{cases} 0 & \text{if } \phi_x(x) \downarrow \\ \uparrow & \text{if } \phi_x(x) \uparrow \end{cases} \tag{2}$$

Here is how to compute ψ:

- Given x, y, ignore y.
- Call $\phi_x(x)$ —that is, $U^{(P)}(x, x)$,
- *If the call ever returns*, then print "0" and halt everything.
- *If it never returns*, then this agrees with the specified in (2) behaviour for $\psi(x, y)$.

By CT, ψ is in $\mathcal{P}$, so, by the S-m-n Theorem, there is a recursive h such that

$$\psi(x, y) = \phi_{h(x)}(y), \text{ for all } x, y$$

You may NOT use S-m-n UNTIL after you have proved that your "$\lambda xy.\psi(x, y)$" is in $\mathcal{P}$.

We can rewrite this as,

$$\phi_{h(x)}(y) = \begin{cases} 0 & \text{if } \phi_x(x) \downarrow \\ \uparrow & \text{if } \phi_x(x) \uparrow \end{cases} \tag{3}$$

or, rewriting (3) *without arguments* (as *equality of functions*, not equality of function calls)

$$\phi_{h(x)} = \begin{cases} \lambda y.0 & \text{if } \phi_x(x) \downarrow \\ \emptyset & \text{if } \phi_x(x) \uparrow \quad \text{says } x \in \overline{K} \end{cases} \tag{3'}$$

In (3′), $\emptyset$ stands for $\lambda y. \uparrow$, the empty function.

Thus,

$$h(x) \in A \text{ iff } \operatorname{ran}(\phi_{h(x)}) = \emptyset \quad \overbrace{\text{iff}}^{\text{bottom case in } 3'} \quad \phi_x(x) \uparrow$$

The above says $x \in \overline{K} \equiv h(x) \in A$, hence $\overline{K} \leq^h A$, and thus $A \notin \mathcal{P}_*$ by Corollary 0.2.4, part ii. □

$$K \stackrel{Def}{=} \{x : \phi_x(x) \downarrow\}$$

$$\overline{K} \stackrel{Def}{=} \{x : \phi_x(x) \uparrow\}$$

Lecture #15, Nov. 9.

0.2.8 Example. The set $B = \{x : \phi_x \text{ has finite domain}\}$ is not semi-recursive.

This is really easy (once we have done the previous example)! All we have to do is "talk about" our findings, above, differently!

We use the same ψ as in the previous example, as well as the same h as above, obtained by S-m-n.

Looking at $(3')$ above we see that the top case has infinite domain, while the bottom one has finite domain (indeed, empty). Thus,

$$h(x) \in B \text{ iff } \phi_{h(x)} \text{ has finite domain} \overbrace{\text{iff}}^{\text{bottom case in } 3'} \phi_x(x) \uparrow$$

The above says $x \in \overline{K} \equiv h(x) \in B$, hence $\overline{K} \leq B$, hence $B \notin \mathcal{P}_*$ by Corollary 0.2.4, part ii. □

0.2.9 Example. Let us mine twice more $(3')$ to obtain two more important undecidability results.

1. Show that $G = \{x : \phi_x \text{ is a constant function}\}$ is undecidable.

   We (re-)use $(3')$ of 0.2.7. Note that in $(3')$ the top case defines a constant function, but the bottom case defines a non-constant. Thus

   $$h(x) \in G \equiv \phi_{h(x)} = \lambda y.0 \equiv \text{top case in } 3' \equiv x \in K$$

   Hence $K \leq G$, therefore $G \notin \mathcal{R}_*$.

2. Show that $I = \{x : \phi_x \in \mathcal{R}\}$ is undecidable. Again, we retell what we can read from $(3')$ in words that are relevant to the set I:

   $$h(x) \in I \overset{\emptyset \notin \mathcal{R}!}{\equiv} \phi_{h(x)} = \lambda y.0 \equiv x \in K$$

   Thus $K \leq I$, therefore $I \notin \mathcal{R}_*$. $\square$

In Notes #8 we will sharpen the result 2 of the previous example.

0.2.10 Example. (The Equivalence Problem, again) We now revisit the equivalence problem and show it is worse than unsolvable (cf. Notes #6):

The relation $\phi_x = \phi_y$ is not semi-decidable.

By 0.2.2, if the 2-variable predicate above is in $\mathcal{P}_*$ then so is $\lambda x.\phi_x = \phi_y$, i.e., taking a constant for y.

Choose then for y a ϕ-index for the *empty function*.

In short,

If the equivalence problem is VERIFIABLE, then so is

$$\phi_x = \emptyset$$

$$Eq = \{x : \phi_x = \emptyset\} = \{x : \mathrm{ran}(\phi_x) = \emptyset\} = A$$

which says the same thing as

$$\mathrm{ran}(\phi_x) = \emptyset$$

We saw that this NOT SEMI-RECURSIVE in 0.2.7. □

0.2.11 Example. *The set $C = \{x : ran(\phi_x)$ is finite$\}$ is not semi-decidable.*

Here we cannot reuse $(3')$ above, because **both** cases in the definition by cases —top and bottom— have functions of ***finite range***.

We want *one* case to have a function of finite range, but the *other* to have infinite range.

Aha! This motivates us to choose a different "ψ" (hence a different "h"), and retrace the steps we took above.

OK, define

$$g(x, y) = \begin{cases} y & \text{if } \phi_x(x) \downarrow \\ \uparrow & \text{if } \phi_x(x) \uparrow \end{cases} \tag{ii}$$

Here is an algorithm for g:

- Given x, y.
- Call $\phi_x(x)$ —i.e., call $U^{(P)}(x, x)$.
- *If this ever returns, then print* "y" and halt everything.
- *If it never returns* from the call, this is the correct behaviour for $g(x, y)$ as well:

  namely, we want $g(x, y) \uparrow$ if $x \in \overline{K}$.

By CT, g is partial recursive, thus by S-m-n, for some recursive unary k we have

$$g(x, y) = \phi_{k(x)}(y), \text{ for all } x, y$$

Thus, by (ii)

$$\phi_{k(x)} = \begin{cases} \lambda y.y & \text{if } x \in K \\ \emptyset & \text{othw} \end{cases} \tag{iii}$$

Hence,

$$k(x) \in C \text{ iff } \phi_{k(x)} \text{ has finite range } \overbrace{\text{iff}}^{\text{bottom case in } iii} x \in \overline{K}$$

That is, $\overline{K} \leq C$ and we are done. □

0.2.12 Exercise. Show that $D = \{x : \mathrm{ran}(\phi_x) \text{ is infinite}\}$ is undecidable. $\square$

0.2.13 Exercise. Show that $F = \{x : \mathrm{dom}(\phi_x) \text{ is infinite}\}$ is undecidable. $\square$

Enough “negativity”!

Here is an important “positive result” that helps to prove that certain relations *ARE* semi-decidable:

0.2.14 Theorem. (Projection theorem; Part I) *A relation $Q(\vec{x}_n)$ that is expressible as*

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

where $S(y, \vec{x}_n)$ is recursive is itself semi-recursive.

Q is obtained by “projecting” S along the y-co-ordinate, hence the name of the theorem.

Proof. Let $S \in \mathcal{R}_*$, and Q be connected as in (1) of the theorem.

Clearly,

$$(\exists y)S(y, \vec{x}_n) \equiv (\mu y)S(y, \vec{x}_n) \downarrow \tag{2}$$

and we know that

$$(\mu y)S(y, \vec{x}_n) \stackrel{Def}{=} (\mu y)c_S(y, \vec{x}_n), \text{ for all } \vec{x}_n \tag{3}$$

Thus $\lambda\vec{x}_n.(\mu y)c_S(y, \vec{x}_n)$ is partial recursive by closure of $\mathcal{P}$ under *UNbounded* search. *Thus so is $\lambda\vec{x}_n(\mu y)S(y, \vec{x}_n)$ by (3).*

Now (1) and (2) give

$$Q(\vec{x}_n) \equiv (\mu y)S(y, \vec{x}_n) \downarrow$$

We are done by Def. 0.1.1. □

0.2.15 Example. The set $A = \{(x, y, z) : \phi_x(y) = z\}$ is semi-recursive.

Here is a verifier for the above predicate:

Given input x, y, z. **Comment**. Note that $\phi_x(y) = z$ is true iff two things happen: (1) $\phi_x(y)\downarrow$ **and** (2) the computed value is z.

1. Given x, y, z.
2. Call $\phi_x(y) = U^{(P)}(x, y)$.
3. If the call returns, then
   - If the output of $U^{(P)}(x, y)$ equals z, then halt everything (the "yes" output).
   - If the output of $U^{(P)}(x, y)$ does NOT equal z, then get into an infinite loop (the "no" case).
4. If the $U^{(P)}(x, y)\uparrow$, *then keep looping* (say "no", by looping).

By CT the above informal verifier can be formalised as a URM M. □

Lecture #16, Nov. 11.

0.3. Projection Theorem II

This section provides a new powerful tool AND proves the converse of Projection Theorem Part I.

How can we trace a (computation of a) URM ?

Exactly in the same manner that we learnt to trace a commercially available program such as C.

0.3.1. Computation simulating functions

Given a URM $M_{X_1}^{\vec{X}_m}$ where —without loss of generality— we selected X_1 as the output variable.

Let all its variables be

$$\overbrace{X_1, \ldots, X_m}^{inputs}, \overbrace{X_{m+1}, \ldots, X_n}^{Non\ inputs} \tag{1}$$

For *any input* $\vec{a}_m$, M's computation can be tabulated in a (*potentially infinite*) table —p.29 below— where for each $y \geq 0$, row y contains the *values* of *ALL the variables in (1)* as well the value of the **Instruction Pointer** IP —that points to the *CURRENT* instruction— at step y.

A "step" is the *act* of executing ONE instruction of M and reaching the next *CURRENT* instruction.

At step zero, ($y = 0$) the computation *ponders the first instruction* of M after the *"I/O Agent" initialised* the input variables and has set all non-input variables to zero.

The entries on the zeroth row are self-evident.

Table 1: M Simulation Table

y	IP	X_1	X_2	$\ldots$	X_m	X_{m+1}	X_{m+2}	$\ldots$	X_n
0	1	a_1	a_2	$\ldots$	a_m	0	0	$\ldots$	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ldots$	$\vdots$	$\vdots$	$\vdots$	$\ldots$	$\vdots$
i	L	b_1	b_2	$\ldots$	b_m	b_{m+1}	b_{m+2}	$\ldots$	b_n
$i+1$	L'	b'_1	b'_2	$\ldots$	b'_m	b'_{m+1}	b'_{m+2}	$\ldots$	b'_n
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ldots$	$\vdots$	$\vdots$	$\vdots$	$\ldots$	$\vdots$

The process for filling the table is algorithmic as follows:

Going *from row* i to *row* $i+1$ (Cf. p.10 in `http://www.cs.yorku.ca/~gt/papers/NOTES-2-URMs.pdf`):

1. L points to $X_k \leftarrow r$. Then $b'_k = r$ while $b'_j = b_j$ for $j \neq k$. Also $L' = L + 1$.

2. L points to $X_k \leftarrow X_k + 1$. Then $b'_k = b_k + 1$ while $b'_j = b_j$ for $j \neq k$. Moreover $L' = L + 1$.

3. L points to $X_k \leftarrow X_k \dot{-} 1$. Then $b'_k = b_k \dot{-} 1$ while $b'_j = b_j$ for $j \neq k$. Moreover $L' = L + 1$.

4. L points to **stop**. Then $b'_j = b_j$ *for all* $j \neq n$. Moreover $L' = L$.

5. L points to **if** $X_k = 0$ **goto** R **else goto** Q. Then $b'_j = b_j$ *for all* $j \neq n$. Moreover $L' = if\ b_k = 0\ then\ R\ else\ Q$.

Note that at "time" y each X_j *and the IP function* hold a value that depends on the initial $\vec{a}_m$ —and on y.

►. Thus we associate with each X_j and with the IP a *TOTAL function* —$\lambda y\vec{a}_m.f_j(y, \vec{a}_m)$ and $\lambda y\vec{a}_m.IP(y, \vec{a}_m)$.

Since I can produce each such function-value, for the X_j and IP —for example, *by hand*— in the *mechanical way* indicated,

by CT, each such function f_j and IP is *RECURSIVE*.

In particular, $f_1 = M_{X_1}^{\vec{X}_m} \in \mathcal{R}$ *and* $\lambda y\vec{a}_m.IP(y, \vec{a}_m) \in \mathcal{R}$.

Important!

0.3.1 Theorem. *With reference to the URM M that we "traced" above, we have that $M_{X_1}^{\vec{X}_m}$ halts for input $\vec{a}_m$ iff there is some step value y where M makes its* ***stop*** *instruction current.*

That is

$\boxed{(\exists y)IP(y, \vec{a}_m) = k}$, *where k is the label of* **stop**

0.3.2 Theorem. (Projection Theorem Part II) *IF* $Q(\vec{x}_m)$ *is semi-recursive,* THEN *there is a recursive* $P(y, \vec{x}_m)$ *such that*

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

Proof. By Definition 0.1.1,

$$Q(\vec{a}_m) \equiv g(\vec{a}_m) \downarrow$$

where $g \in \mathcal{P}$.

Let then $g = M_{X_1}^{\vec{X}_m}$.

By 0.3.1,

$g(\vec{a}_m) \downarrow \equiv (\exists y) IP(y, \vec{a}_m) = q$, where q labels **stop** in M

But $\underline{IP(y, \vec{a}_m) = q \text{ is recursive}}$ so we may *take it as the "$P(y, \vec{x}_m)$" we want.* $\square$