Summary

Definitions ... 2

- Reduction ... 2
- Recursive Set ... 2
- Recursively Enumerable Set ... 2
- Decidable Predicate ... 2
- Semi-decidable predicate ... 2
- Structure Theorem ... 3
- Projection Theorem ... 4
- Primitive Recursive Functions ... 4
- Smn-Theorem ... 4
- Saturated set ... 4
- Rice's Theorem ... 5
- Rice-Shapiro's Theorem ... 5
- Second Recursion Theorem ... 5

Exercises ... 6

- URM-Machines ... 6
- Smn-theorem ... 8
- Primitive recursive functions ... 9
- Functions and computability ... 11
  - Diagonalization ... 11
- Recursiveness of sets ... 13
  - Rice-Shapiro ... 13
  - Reduction ... 15
- Second Recursion Theorem ... 16
  - Show there exist an index s.t. function is total/computable ... 16
  - Show there exist an index s.t. function is not computable ... 16
  - Show that a set A is not saturated ... 16

Definitions

Reduction

DEFINITION 13.5. Let $A, B \subseteq \mathbb{N}$. We say that the problem $x \in A$ *reduces* to the problem $x \in B$ (or simply that A reduces to B), written $A \leqslant_m B$ if there exists a function $f : \mathbb{N} \to \mathbb{N}$ computable and total such that, for every $x \in \mathbb{N}$

$$x \in A \quad \Leftrightarrow \quad f(x) \in B$$

In this case, we say that f is the *reduction function.*

Recursive Set

DEFINITION 13.1. A set $A \subseteq \mathbb{N}$ is *recursive* if its characteristic function

$$\chi_A : \mathbb{N} \to \mathbb{N}$$

$$\chi_A(x) = \begin{cases} 1 & x \in A \\ 0 & x \notin A \end{cases}$$

is computable.

Recursively Enumerable Set

DEFINITION 15.1 (Recursively enumerable set). We say that $A \subseteq \mathbb{N}$ is *recursively enumerable* if the semi-characteristic function

$$sc_A(x) = \begin{cases} 1 & x \in A \\ \uparrow & \text{otherwise} \end{cases}$$

is computable.

Specifically:

- A set is r.e. if I can check a property on a finite number of points
- A set is not r.e. if I have to check the property on an infinite number of points

Decidable Predicate

A predicate $Q(\vec{x}) \subseteq \mathbb{N}^k$ is decidable if the characteristic function $\chi_Q : \mathbb{N}^k \to \mathbb{N}$ defined by

$$\chi_Q(\vec{x}) = \begin{cases} 1 & \text{if } Q(\vec{x}) \\ 0 & \text{otherwise} \end{cases}$$

is computable.

Semi-decidable predicate

A predicate $Q(\vec{x}) \subseteq \mathbb{N}^k$ is semi-decidable if the semi-characteristic function sc_Q $\mathbb{N}^k \to \mathbb{N}$ defined by

$$sc_Q(\vec{x}) = \begin{cases} 1 & \text{if } Q(\vec{x}) \\ \uparrow & \text{otherwise} \end{cases}$$

is computable.

Structure Theorem

Let $P(\vec{x}) \subseteq \mathbb{N}^k$ a predicate. Then $P(\vec{x})$ is semidecidable iff there is a dedicable predicate $Q(t, \vec{x}) \subseteq \mathbb{N}^{k+1}$ s.t. $P(\vec{x}) = \exists t. Q(t, \vec{x})$

Let $P(\vec{x}) \subseteq \mathbb{N}^K$ a predicate

$P(\vec{x})$ semi-decidable $\iff$ there is $Q(t, \vec{x}) \subseteq \mathbb{N}^{K+1}$ decidable s.t. $P(\vec{x}) = \exists t. Q(t, \vec{x})$

<u>Note:</u> in the "notes.pdf" the predicate is written as "decidable", but prof. says it's semidecidable like written here. Keep this in mind.

This reasoning is useful inside theoretical exercises about decidability/semidecidability because it's literally the same reasoning, reported here for the sake of completeness.

PROOF. $(\Rightarrow)$ Let $P(\vec{x})$ be semi-decidable. It has a computable semi characteristic function sc_P so

$$P(\vec{x}) \equiv \exists t. H(e, \vec{x}, t)$$

therefore if we can rewrite H as $Q(t, \vec{x}) = H(e, \vec{x}, t)$, in this way Q is decidable as we wanted and

$$P(\vec{x}) \equiv \exists t. Q(t, \vec{x})$$

$(\Leftarrow)$ Let $P(\vec{x}) \equiv \exists t. Q(t, \vec{x})$ with $Q(t, \vec{x})$ decidable. Observe that

$$sc_P(\vec{x}) = \mathbf{1}(\mu t. |\chi_Q(t, \vec{x}) - 1|)$$

which is computable by definition, and therefore $P(\vec{x})$ is semi-decidable.

The converse does not hold, for example $P(\vec{x}, y) \equiv (x \in W_x)\ \exists x. P(x, \vec{y})$

Alternatively:

- $P(x, y) \equiv (y = 1) \land (x \notin W_x) Q(y) \equiv \exists x. P(x, y) \equiv (y = 1)$
- Suppose $P(x, y)$ holds if $\phi_x(x) \uparrow, P(x, y)$ non-semi-decidable, otherwise $\overline{K}$ would be r.e.. We know there are programs inside $\overline{K}$, e.g. the ones calculating the always undefined function, but then $\exists x. P(\vec{x}, y)$ always holds and so it would always be inevitably undecidable

Projection Theorem

THEOREM 15.6 (Projection theorem). *Let $P(x, \vec{y})$ be semi-decidable; then*

$$\exists x.P(x, \vec{y}) = P'(\vec{y})$$

is semi-decidable.

This reasoning is useful inside theoretical exercises about decidability/semidecidability because it's literally the same reasoning, reported here for the sake of completeness.

Proof

Let $P(x, \vec{y}) \subseteq \mathbb{N}^{k+1}$ semi-decidable. Hence, by the structure theorem, there is $Q(t, x, \vec{y}) \subseteq \mathbb{N}^{k+2}$ decidable s.t. $P(x, \vec{y}) \equiv \exists t. Q(t, x, \vec{y})$.

Now $R(\vec{y}) \equiv \exists x. P(x, \vec{y}) \equiv \exists x. \exists t. Q(t, x, \vec{y}) \equiv \exists w. Q((w)_1, (w)_2, \vec{y})$ is decidable.

Hence, R is the existential quantification of a decidable predicate and by the structure theorem is semi-decidable.

Primitive Recursive Functions

Solution: The set $\mathcal{PR}$ of primitive recursive functions is the smallest set of functions that contains the basic functions:

1. $\mathbf{0} : \mathbb{N} \to \mathbb{N}$defined by $\mathbf{0}(x) = 0$ for each $x \in \mathbb{N}$;
2. $\mathbf{s} : \mathbb{N} \to \mathbb{N}$defined by $\mathbf{s}(x) = x + 1$ for each $x \in \mathbb{N}$;
3. $\mathbf{U}_j^k : \mathbb{N}^k \to \mathbb{N}$defined by $\mathbf{U}_j^k(x_1, \dots, x_k) = x_j$ for each $(x_1, \dots, x_k) \in \mathbb{N}^k$.

and which is closed with respect to generalized composition and primitive recursion, defined as follows. Given the functions $f_1, \dots, f_n : \mathbb{N}^k \to \mathbb{N}$and $g : \mathbb{N}^n \to \mathbb{N}$ their generalized composition is the function $h : \mathbb{N}^k \to \mathbb{N}$ defined by:

$$h(\vec{x}) = g(f_1(\vec{x}), \dots, f_n(\vec{x})).$$

Given the functions $f : \mathbb{N}^k \to \mathbb{N}$ and $g : \mathbb{N}^{k+2} \to \mathbb{N}$ the function defined by primitive recursion is $h : \mathbb{N}^{k+1} \to \mathbb{N}$:

$$\begin{cases} h(\vec{x}, 0) = f(\vec{x}) \\ h(\vec{x}, y+1) = g(\vec{x}, y, h(\vec{x}, y)) \end{cases}$$

Smn-Theorem

Given $m, n \geq 1$ there is a total computable function $s_{m,n}: \mathbb{N}^{m+1} \to \mathbb{N}$ s.t. $\forall \vec{x} \in \mathbb{N}^m, \forall \vec{y} \in \mathbb{N}^n, \forall e \in \mathbb{N}$

$$\phi_e^{(m+n)}(\vec{x}, \vec{y}) = \phi_{s_{m,n}(e,\vec{x})}^{(n)}(\overrightarrow{y})$$

Saturated set

A set $A \subseteq \mathbb{N}$ is saturated whenever, if it includes the index (program) for a computable function, it includes also all the other indexes (programs) for the same function. Formally, for all $x, y \in \mathbb{N}$ if $x \in A$ and $\varphi_x = \varphi_y$ then $y \in A$.

Rice's Theorem

THEOREM 14.6 (Rice's theorem). *Let $A \in \mathbb{N}, A \neq \varnothing, A \neq \mathbb{N}$ be saturated. Then it is not recursive.*

Rice-Shapiro's Theorem

Let $A \subseteq C$ (where A is a property of functions) be a set of computable functions and let $A = \{x \mid \phi_x \in \mathrm{A}\}$ Then if A is r.e. then

$$\forall f \ (f \in A \Leftrightarrow \exists \theta \subseteq f, \theta \text{ finite s.t. } \theta \in A)$$

Generally, it can be used in two ways:

- $\exists f \in C. f \notin A \wedge \exists \theta \subseteq f \ finite, \theta \in A \Rightarrow \mathrm{A} \ not \ r.e.$
- $\exists f \in C. f \in A \wedge \forall \theta \subseteq f \ finite, \theta \notin A \Rightarrow \mathrm{A} \ not \ r.e.$

Second Recursion Theorem

The Second Recursion Theorem says that: for all functions $f : \mathbb{N} \to \mathbb{N}$, if f is total and computable then there is $e \in \mathbb{N}$ such that $\varphi_e = \varphi_{f(e)}$.

Exercises

URM-Machines

This kind of exercises was mainly present only inside partial exams.

- The exercise gives us a variant of the normal URM model which these basic instructions:

    - *zero* $Z(n)$, which sets the content of register R_n to zero: $r_n \leftarrow 0$
    - *successor* $S(n)$, which increments by 1 the content of register R_n: $r_n \leftarrow r_n + 1$
    - *transfer* $T(m, n)$, which transfers the content of register R_m into R_n, which R_m staying untouched: $r_n \leftarrow r_m$
    - *conditional jump*: $J(m, n, t)$, which compares the content of register R_m and R_n, so:
        - if $r_m = r_n$ then jumps to I_t (jumps to t-th instruction)
        - otherwise, it will continue with the next instruction

- We have to prove the inclusion of the computable sets in both ways
    - From modified URM to normal URM
    - From normal URM to modified URM

- Define $\mathcal{C}$ for URM-machine and $\mathcal{C}'$ (for example) the set of the model you have to show

- First step is showing $\mathcal{C}' \subseteq \mathcal{C}$
    - Not necessarily the new machine is more powerful, infact it may be even less powerful

    - Informally, we simply can code the "new" instruction/s in normal URM machine using a routine of some existing instructions (jump/transfer/successor/jump)
        - This is typically done considering say i the index of an unused register by the program and a subroutine

    - Formally, we prove $\mathcal{C}' \subseteq \mathcal{C}$ showing that, for each number of arguments k and for each program P using both sets of instructions we can obtain a URM program P' which computes the same function i.e. such that $f_{P'}^{\prime(k)} = f_P^{(k)}$

    - The proof goes on by induction on the number of instructions h
        - ($h = 0$), usually trivial, it's already a URM program

        - ($h \to h + 1$), basically I will describe the logic
            - Describe as j for instance the index of instruction you want to replace and $l(P)$ the length of computed program
            - We can build a program P'' using a register not referenced in P, for instance $q = \max\{\rho(P), k\} + 1$ (ρ is the largest unused register)
            - Show that for the whole length of program, the jump to the subroutine can successfully replace the instruction wanted

        - The program P'' is s.t. $f_{P''}^{(k)} = f_P^{(k)}$ and it contains h instructions. By inductive hypothesis, there exists a URM program P' s.t. $f_{P'}^{(k)} = f_P^{(k)}$, which is the desired program

- Second step is showing $\mathcal{C} \subseteq \mathcal{C}'$
  - The usual question is if inclusion holds both ways or if it is strict
  - If this second part does not hold, then it is not strict

- Usually, this is similar to the one before, but this time around, instructions of normal URM have to be encoded using only the new machine
  - This one follows, if formally, exactly the same steps as before

Smn-theorem

- Give a function of two arguments $g(x, y)$
  - Define a case for set definition
  - Define a case for otherwise
- In this case, with smn-theorem exercises, it helps creating a function s.t.
  - the domain is where the values exist
    - so, the positive case condition is the domain or less than the domain and has to include that case inside condition
  - the codomain is the output we want to reach
    - after having written the cases, we see if the output/the computable function respects said condition
- It is computable, since it is defined by cases
- By the smn-theorem, there is $s: \mathbb{N} \to \mathbb{N}\ s.t. \forall x, y \in \mathbb{N}$
  - Write $\phi_{s(x)}(y) = g(x, y)$ and rewrite the function defined initially again
- As observed above
  - $W_{s(x)}$ = domain given by definition
    - $W_x = \{y \mid g(x, y) \downarrow\}$
  - $E_{s(x)}$ = codomain given by definition
    - $E_x = \{g(x, y) \mid condition\ of\ defined\ case)$

In case you have $E_{k(n)}$ and $W_{k(n)}$ inside the function definition (just notation here, folks, the concept holds the same way, you simply have n in place of x):

- simply use a function $f(n, x)$
- by the smn theorem, there is a total computable function $k: \mathbb{N} \to \mathbb{N}\ s.t. \phi_{k(n)}(x) = f(n, x)\ \forall n, x \in \mathbb{N}$
- As observed above
  - $W_{k(x)}$ = domain given by definition
  - $E_{k(x)}$ = codomain given by definition

Primitive recursive functions

- Write the $\mathbb{PR}$ class definition present above
- Carefully read the problem definition and write it using a combination of known functions

Consider, just for reference, these basic functions are primitive recursive functions:

- *sum* $x + y$

$$h: N^2 \to N, h(x,y) = x + y$$

$$\begin{cases} h(x,0) = x = f(x) \\ h(x,y+1) = h(x,y) + 1 = g(h(x,y)) \end{cases}$$

$$f(x) = x$$
$$g(x,y,z) = z + 1$$

- *product* $x * y$

$$h': N^2 \to N, h'(x,y) = x * y$$

$$x \cdot 0 = 0$$
$$x \cdot (y+1) = (x \cdot y) + x$$

$$f(x) = 0$$
$$g(x,y,z) = z + y$$

- *exponential* x^y

$$\begin{array}{lll} x^0 = 1 & h(x,0) = 1 & f(x) = 1 \\ x^{y+1} = x^y \cdot x & h(x,y+1) = h(x,y) \cdot x & g(x,y,z) = z \cdot x \end{array}$$

- *predecessor* $y - 1$

$$\begin{array}{lll} 0 \dot{-} 1 = 0 & h(0) = 0 & f \equiv \underline{0} \\ (x+1) \dot{-} 1 = x & h(x+1) = x & g(y,z) = y \end{array}$$

- *difference* $x \dot{-} y = \begin{cases} x - y & x \geqslant y \\ 0 & \text{otherwise} \end{cases}$

$$\begin{array}{ll} x \dot{-} 0 = x & f(x) = x \\ x \dot{-} (y+1) = (x \dot{-} y) \dot{-} 1 & g(x,y,z) = z \dot{-} 1 \end{array}$$

- *sign* $sg(x) = \begin{cases} 0 & x = 0 \\ 1 & x > 0 \end{cases}$

$$\begin{array}{ll} sg(0) = 0 & f \equiv \underline{0} \\ sg(x+1) = 1 & g(y,z) = 1 \end{array}$$

- *negative sign* (or *complement sign*)

$$\bar{sg}(x) = \begin{cases} 1 & x = 0 \\ 0 & x > 0 \end{cases}$$

$$\begin{array}{ll} sg(0) = 0 & f \equiv \underline{0} \\ sg(x+1) = 1 & g(y,z) = 1 \end{array}$$

- *minimum*

$$min(x,y) = x \dot{-} (x \dot{-} y);$$

- *maximum*

$$max(x,y) = (x \dot{-} y) + y;$$

- *remainder*

$$rm(x,y) = \begin{cases} y \mod x & x \neq 0 \\ y & x = 0 \end{cases}$$

$$rm(x,0) = 0$$

$$rm(x,y+1) = \begin{cases} rm(x,y)+1 & rm(x,y)+1 \neq x \\ 0 & \text{otherwise} \end{cases}$$

$$= (rm(x,y)+1) \cdot sg((x \dot{-} 1) \dot{-} rm(x,y))$$

- *quotient*, $qt(x,y) = y \text{ div } x$ (convention $qt(0,y) = y$), we define:

$$qt(x,y+1) = \begin{cases} qt(x,y)+1 & rm(x,y)+1 = x \\ qt(x,y) & \text{otherwise} \end{cases}$$
$$= qt(x,y) + sg((x \dot{-} 1) \dot{-} rm(x,y))$$

For completeness sake, always assume the sum and product are bounded, so they are primitive recursive (bounded sum and product)

- You define the function of the problem as a combination of base case and recursive case of the base functions and also some like the ones presented here

Functions and computability

- In this case, consider the function are total
    - So, they have to define and handle all cases by definition

We have different choices to follow:

- diagonalization (subsection ahead)
- use a known non computable function, like χ_K
    - conditions are dependent on exercise, here reported just as an example

$$f(x) = \begin{cases} 0, & if\ x \leq 1 \\ \chi_K(x), & otherwise \end{cases}$$

    - the general structure would be using χ_K somewhere, it can be both on positive/otherwise case
- sometimes, it happens that we use functions and subfunctions

$$\theta(x) = \begin{cases} f(x), & general\ condition\ (e.g. if\ x < x_0) \\ \uparrow, & otherwise \end{cases}$$

$$f(x) = \begin{cases} \theta(x), & general\ condition\ (e.g. if\ x < x_0) \\ value\ (e.g.\ \ 0, k), & otherwise \end{cases}$$

- since the subfunction is finite, the function is too, and one can write it as a computable function

Diagonalization

- In this case, there are notable total non-computable functions; the function is built to differ from its own values by recursion
- We then say $f(x) \neq \phi_x(x)$ since this holds by construction (just use the problem conditions replacing $f(x)$ with $\phi_x(x)$)

Consider (conditions are dependent on exercise, here reported just as an example):

$$g(x) = \begin{cases} \phi_x(x) + 1, & x \in W_x \\ 0, & otherwise \end{cases}$$

More generally, it might be something like:

$$f(x) = \begin{cases} something\ involving\ \phi_x(x), & x \in W_x \\ 0, & otherwise\ (so, x \notin W_x) \end{cases}$$

The good proof (extended by this) would be:

- f is total by construction
- f is not computable since $\forall x \in N, f(x) \neq \phi_x(x)$
    - infact, if $\phi_x(x) \downarrow$ then $f(x) \neq something\ involving\ \phi_x(x) \neq \phi_x(x)$
    - if $\phi_x(x) \uparrow$ then $f(x) = 0 \neq \phi_x(x)$
- the specified exercise property holds

- Consider the following notable examples from the course:

OBSERVATION 10.4. There exists a total non-computable function $f : \mathbb{N} \to \mathbb{N}$ defined by

$$f(n) = \begin{cases} \varphi_n(n) + 1 & \text{if } \varphi_n(n) \downarrow \\ 0 & \text{if } \varphi_n(n) \uparrow \end{cases}$$

f is not computable because it differs from all computable functions. In fact

- if $\varphi_n(n) \downarrow$, then $f(n) = \varphi_n(n) + 1 \neq \varphi_n(n)$
- if $\varphi_n(n) \uparrow$, then $f(n) = 0 \neq \varphi_n(n)$

so

$$\forall n \; f \neq \varphi_n$$

OBSERVATION 10.5. There are infinitely many total non-computable functions of the following shape

$$f(n) = \begin{cases} \varphi_n(n) + k & n \in W_n \\ k & n \notin W_n \end{cases}$$

EXERCISE 10.6. Let $f : \mathbb{N} \to \mathbb{N}$, $m \in \mathbb{N}$. Show that there exists a non-computable function $g : \mathbb{N} \to \mathbb{N}$ such that

$$g(x) = f(x) \quad \forall x < m$$

Idea: use a "translated diagonal":

$$g(x) = \begin{cases} f(x) & x < m \\ \varphi_{x-m}(x) + 1 & x \geqslant m \text{ and } x \in W_{x-m} \\ 0 & x \geqslant m \text{ and } x \notin W_{x-m} \end{cases}$$

g is not computable since $g(x + m) \neq \varphi_x(x + m)$ for all x, so

$$\forall x \; g \neq \varphi_x$$

Another approach is to define g in the following way

$$g(x) = \begin{cases} f(x) & x < m \\ \varphi_x(x) + 1 & x \geqslant m \text{ and } x \in W_x \\ 0 & x \geqslant m \text{ and } x \notin W_x \end{cases}$$

because each function appears infinitely many times in the enumeration, and skipping the first $m - 1$ steps does not create any problem. Formally

$$\forall x \geqslant m \quad g \neq \varphi_x$$

so for all y

$$\forall y \; \exists x \geqslant m \; \varphi_y = \varphi_x$$

- Other cases, more similar to what we saw in the course, involve multiple cases, usually three, with small variations of the condition but with the same concept

$$f(x) = \begin{cases} \frac{x}{2}, & x \; even \\ \phi_{\frac{x-1}{2}}(x) + 1, & if \; x \; odd \; and \; x \in W_{\frac{x-1}{2}} \\ 0, & otherwise \end{cases}$$

The same observations about using ϕ_x hold.

Written by Gabriel R.

Recursiveness of sets

Rice-Shapiro

- We use this one if A is saturated
  - This usually happens when the exercises gives W_x, E_x or both of them
  - A set is saturated if there is a non-trivial property (finitely characterizable)
  - $A = \{x \in \mathbb{N} \mid \phi_x \in \mathcal{A}\}$ and $\mathcal{A} = \{f \mid \ldots\}$
  - You replace W_x with $dom(f)$ and E_x with $cod(f)$ (also $\phi_x(x)$ with $f(x)$)

- This way, we show A and $\overline{A}$ are not r.e.
  - This may not always be the case; sometimes a set is saturated, but the set is r.e. (it means you can write a semicharacteristic function χ_A)
    - In this case, if A is r.e. $\overline{A}$ is not r.e (hence not recursive)
    - Conversely, if $\overline{A}$ is r.e., A not r.e. (hence not recursive)

- Applying the definition it means either:
  - we have a function which is in the set but a finite subfunction not in the set
  - we have a function which is not in the set but a finite subfunction which is in the set

- Usually, we use id and $\emptyset$
  - identity = defined for all natural numbers
    - if you use this one, possibly you have a function inside/not inside the set
  - always undefined function = undefined for all natural numbers
    - this one is often used as a subfunction to prove is inside the complement
    - many other times, it can simply be a function inside the normal set
- Sometimes, one can use the constant $\mathbf{1}$ function (or constant 0)

- It usually works showing you have (as above, but replace f with a logically correlated function to the exercise definition of specified set)
  - $f \notin \mathcal{A}$, but $\theta \in \mathcal{A}$
  - $f \in \mathcal{A}$, but $\theta \notin \mathcal{A}$
- This usually holds for both sets
  - If both sets are not r.e. they are not recursive either

There are the following implications:

- if A is r.e. but not recursive, also $\overline{A}$ is not r.e. (also not recursive, otherwise they would be both recursive)
- if A is recursive, then χ_A is computable. We have $\overline{A}$ is r.e. and:
  - if $K \leq_m \overline{A}$, then $\overline{A}$ is not recursive
  - if $\chi_{\overline{A}}$ is computable then $\overline{A}$ is recursive
- If A r.e., then $\overline{A}$ is not – if A is r.e., it means sc_A exists, but is not recursive
- If $\overline{A}$ r.e. then A is not – if $\overline{A}$ is r.e., it means $sc_{\overline{A}}$ exists, but is not recursive

Side note (important):

- One can show a set is not recursive by using Rice's theorem
    - This occurs when the set is saturated and maybe is r.e. but we ask if it is recursive
    - Then, you use $e_0 \in id/\mathbf{1}$ and $e_i \in \emptyset$ to prove $e_0 \in A, e_1 \notin A$ hence $A \neq \emptyset, \mathbb{N}$
        - for example $e_0\ s.t.\ \phi_{e_0} = id/\mathbf{1}$ or $e_1 s.t.\ \phi_{e_1} = \emptyset$

Usually, if the set is not r.e. it is also not recursive.

Reduction

To note:

- $K \leq_m A$: to prove a set is not recursive
- $\overline{K} \leq_m A$: to prove a set is not r.e.

- We use this one if A is not recursive ($K \leq_m A$)
  - usually something like $g(x,y) = \begin{cases} y \ (or\ value), & x \in K \\ \uparrow, & otherwise \end{cases}$
  - a variant with the same meaning is $g(x,y) = \begin{cases} 1 \ (or\ value), & x \in W_x \\ \uparrow, & otherwise \end{cases}$
  - sometimes, consider there is also: $g(x,y) = \begin{cases} \phi_x(x), & x \in W_x \\ \uparrow, & otherwise \end{cases}$
    - this one occurs in case of both domain and codomain over index x
  - it is computable and thus, by the smn theorem, we deduce that there is a total computable function $s: \mathbb{N} \to \mathbb{N}$ such that, for each $x, y \in \mathbb{N}$, $g(x,y) = \phi_{s(x)}(y)$
- It can be shown to be the correct reduction function
  - if $x \in K$, $\phi_{s(x)}(y) = g(x,y) = y \ (or\ value)\ \forall y \in \mathbb{N}$. Therefore $s(x) \in W_{s(x)} = \mathbb{N}$ and $\phi_{s(x)}(s(x)) = s(x)$. Therefore, $s(x) \in A$
    - the function here is the value; if we had y^2 it would have been $(s(x))^2$
  - if $x \notin K$, $\phi_{s(x)}(y) = g(x,y) \uparrow \forall\ y \in N$. Therefore $s(x) \notin W_{s(x)} = \emptyset$ and so $s(x) \notin A$

- Note: if $K <_m A$, then A usually is r.e.

- We can also use the complement of the same set ($\overline{K} \leq_m A$)
  - usually something like $g(x,y) = \begin{cases} y \ (or\ value, usually\ 0), & \neg H(x,x,y) \\ \uparrow, & otherwise \end{cases}$
  - this starts from a computable function, like $g(x,y) = \begin{cases} 1 \ (or\ value), & x \in K \\ \uparrow, & otherwise \end{cases}$
  - it is computable since we have $g(x,y) = value * sc_K(x)$ and thus, by the smn theorem, we deduce that there is a total computable function $s: \mathbb{N} \to \mathbb{N}$ such that, for each $x, y \in \mathbb{N}$, $g(x,y) = \phi_{s(x)}(y)$
- It can be shown to be the correct reduction function
  - if $x \in \overline{K}$, $\phi_{s(x)}(y) = g(x,y) = y \ (or\ value)\ \forall y \in \mathbb{N}$. Also, we can say $H(x,x,y)$ is false $\forall y \in \mathbb{N}$. Therefore $s(x) \in W_{s(x)} = \mathbb{N}$ and $\phi_{s(x)}(s(x)) = s(x)$. Therefore, $s(x) \in A$
  - if $x \notin \overline{K}$, $\phi_{s(x)}(y) = g(x,y) \uparrow \forall\ y \in N$. Also, we can say $H(x,x,y)$ is true $\forall y \in \mathbb{N}$ Therefore $s(x) \notin W_{s(x)} = \emptyset$ and so $s(x) \notin A$

- If this reduction from complement holds, A is not r.e.

- It can also happen $\overline{K} \leq_m \overline{A}$ and so $\overline{A}$ is not r.e.
- If both are valid (so $\overline{K} \leq_m A$ and $\overline{K} \leq_m \overline{A}$), both sets ($A, \overline{A}$) are not r.e.

Second Recursion Theorem

Show there exist an index s.t. function is total/computable

- Give the theorem definition
- Give a function of two arguments $g(x, y)$ for instance defined by cases
    - case for the normal condition
    - case for otherwise
- Since it is defined by cases, it's computable (since it is total, holds)
- By the smn-theorem, there exists a total computable function $s: N \to N\ s.t.\ \phi_{s(x)}(y) = g(x, y)$
- By the Second Recursion Theorem, there exists $e \in \mathbb{N}$ such that $\phi_e = \phi_{s(e)}$
- You use the function previously defined and replace $g(x, y)$ with $\phi_e(y) = \phi_{s(e)}(y) = g(e, y)$
    - inside the function, replace x with e
- You conclude since you fixed the point in which all the conditions you posed hold (simply use second recursion theorem definition)

Show there exist an index s.t. function is not computable

- Give the theorem definition
- Note the function is computable but it is usually total, so you have say $\phi_x \neq \phi_{h(x)}$
- By the Second Recursion Theorem, there exists $e \in \mathbb{N}$ such that $\phi_e \neq \phi_{s(e)}$
- So, the original function cannot be computable

Show that a set A is not saturated

- Give the theorem definition
- Give a function of two arguments $g(x, y)$ for instance defined by cases
    - case for the normal condition
    - case for otherwise
- Since it is defined by cases, it's computable
- By the smn-theorem, there exists a total computable function $s: N \to N\ s.t.\ \phi_{s(x)}(y) = g(x, y)$
- By the Second Recursion Theorem, there exists e such that $\phi_e = \phi_{s(e)}$
- You use the function previously defined and replace $g(x, y)$ with $\phi_e(y) = \phi_{s(e)}(y) = g(e, y)$
    - inside the function, replace x with e
- Now, just take $e' \neq e$ such that $\phi_e' = \phi_e$ (which exists since there are infinitely many indices for the same computable function)
- So, we have e in A and $e' \notin A$ So, A is not saturated