Summary

[Definitions 2](#_Toc156120623)

[Reduction 2](#_Toc156120624)

[Recursive Set 2](#_Toc156120625)

[Recursively Enumerable Set 2](#_Toc156120626)

[Decidable Predicate 2](#_Toc156120627)

[Semi-decidable predicate 2](#_Toc156120628)

[Primitive Recursive Functions 3](#_Toc156120629)

[Smn-Theorem 3](#_Toc156120630)

[Saturated set 3](#_Toc156120631)

[Rice’s Theorem 3](#_Toc156120632)

[Rice-Shapiro’s Theorem 3](#_Toc156120633)

[Second Recursion Theorem 4](#_Toc156120634)

[Exercises 5](#_Toc156120635)

[Smn-theorem 5](#_Toc156120636)

[Recursiveness of sets 6](#_Toc156120637)

[Rice-Shapiro 6](#_Toc156120638)

[Reduction 7](#_Toc156120639)

[Second Recursion Theorem 8](#_Toc156120640)

[Show that a set A is not saturated 8](#_Toc156120641)

# Definitions

### Reduction

Immagine che contiene testo, Carattere, schermata, bianco

Descrizione generata automaticamente

### Recursive Set

Immagine che contiene testo, Carattere, schermata, linea

Descrizione generata automaticamente

### Recursively Enumerable Set

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

### Decidable Predicate

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

### Semi-decidable predicate

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

### Primitive Recursive Functions

The set of primitive recursive functions is the least class of functions including:

* successor
* zero
* projection

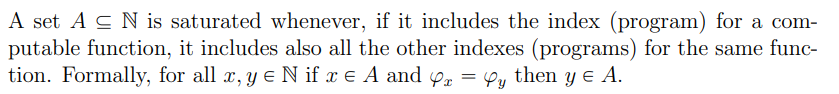
which is closed under:

* generalized composition: given and , their generalized composition is given from function s.t.
* primitive recursion: given , primitive recursion operation is defined as

### Smn-Theorem

Given there is a total computable function such that

### Saturated set



### Rice’s Theorem

Let . If is saturated, then is not recursive.

### Rice-Shapiro’s Theorem

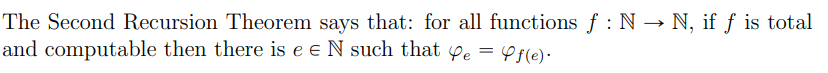
Let (where is a property of functions) be a set of computable functions and let Then if is r.e. then



finite s.t. )

Generally, it can be used in two ways:

### Second Recursion Theorem



# Exercises

## Smn-theorem

* Give a function of two arguments
  + 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
  + Write and rewrite the function defined initially again
* As observed above
  + domain given by definition
  + = codomain given by definition

In case you have and inside the function definition (just notation here, folks, the concept holds the same way, you simply have in place of ):

* simply use a function
* by the smn theorem, there is a total computable function
* As observed above
  + domain given by definition
  + = codomain given by definition

## Primitive recursive functions

* Write the definition conveniently written above

## Decidability and semidecidability

## Diagonalization

## Recursiveness of sets

### Rice-Shapiro

* We use this one if is saturated
  + This usually happens when the exercises gives or both of them
  + and
  + You replace with and with
* This way, we show and 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 )
    - In this case, if is r.e. is not r.e (hence not recursive)
    - Conversely, if is r.e., 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 and
  + identity = defined for all natural numbers
  + always undefined function = undefined for all natural numbers
* Sometimes, one can use the constant function
* It usually works showing you have (as above, but replace with a logically correlated function to the exercise definition of specified set)
  + , but
  + , but
* This usually holds for both sets
  + If both sets are not r.e. they are not recursive either

Side note:

* One can show a set is not recursive by using Rice’s theorem
  + This occurs when the set is saturated
  + Then, you use and to prove hence

### Reduction

* We use this one if is not recursive

## Second Recursion Theorem

### Show that a set A is not saturated

* Give a function of two arguments 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
* By the Second Recursion Theorem, there exists such that and so
* You use the function previously defined and replace with
* inside the function, replace with
* Now, just take such that (which exists since there are infinitely many indices for the same computable function).
* So, we have in and So, is not saturated