*Lessons covered by this meeting according to schedule:*

* 01/10/2024
  + Historical introduction. From logics to computability to modern computer science.
* 07/10/2024
  + Effective procedures and computable functions. Relations, functions, sets and cardinality. Existence of non-computable functions.
* 08/10/2024
  + URM-computability. The class C of URM-computable functions. Examples. [§1.2, §1.3]
* 14/10/2024
  + Exercises on some variants of the URM machine
  + Decidable predicates. [§1.4]
  + Computability on domains different from the natural numbers. [§1.5, §3.6]
* 15/10/2024
  + Generating computable functions: Notation. Closure under generalized composition (Substitution) [§2.1, §2.2, §2.3]
  + Primitive recursion and examples [§2.4]

Actual lesson

* **Existence of computable functions**

Description of effective procedures as sequences of elementary steps transforming input into outputs. Make people think about the sense of functions (mapping/etc.) - § page 52 of my notes.

Notion of total function: domain and codomain/image

* : function domain
  + Letter *probably* stands for *writable*, so it is possible to write down all the inputs for which the function is defined)
* : image of a function (aka, all the function outputs or the values in the codomain – the first definition is less math-like and more human-like, I’d say, but let’s try to be precise)
  + remember the codomain is also written as , where is a function
  + remember also the image can be written as
  + Letter *probably* stands for *enumerable* because it is possible to enumerate all the outputs that the function can produce.

Notion of subtraction (monus) with example

Immagine che contiene testo, Carattere, calligrafia, bianco

Descrizione generata automaticamente

This represent a subtraction which *will never give you negative results*, so it’s always positive and well-defined with no problems (if the subtraction gives say , the calculation will give , something like that).

More precisely:

* the subtraction with a point on top indicates something like a normal subtraction, it saturates to zero when we get a negative value (not like a normal subtraction but defined only for natural numbers). Examples:
* Immagine che contiene testo, schermata, Carattere, algebra

  Descrizione generata automaticamentethis is technically called “truncated subtraction”, as shown [here](https://math.stackexchange.com/questions/328530/is-the-set-of-natural-numbers-closed-under-subtraction)

You can see [here](https://en.wikipedia.org/wiki/Monus#:~:text=Truncated%20subtraction%20is%20usually%20defined,subtraction%203%20%E2%88%B8%205%20%3D%200.) it’s called *monus* (or also *cut-off subtraction* as evidenced by Cutland, p. 241). As you can see above, the function is primitive recursive and usually you put a point because you subtract from the highest value the lowest, so this ensures “it’s all good”.

* **Existence of non-computable function**

Definition of partial function/totality

Total = defined for all natural numbers (*all set*), assigning a unique element to each output from input

* A function is total if it’s defined for cases and returns an output for every single possible input (covers all possibilities, in other terms) – so, a function by cases is total by construction
* Technically, a function is total when is defined for every input on set
* Diagonalization here is a powerful tool
  + We can use diagonalization to make a partial computable function that differs from every total computable function
  + It states that there are sets where you can’t list all of their members sequentially. It assumes to have an infinite list of elements, which can’t be because the underlying thing is total
    - If a list of a set of these strings exists…
    - …then there also exists a string that is not in the list
  + Given it does not happen, there is the contradiction: the function is total, but you find a value which is not on the list, while being defined for all values

Partial: defined *only on a subset* of a specified set. Consider a partial function from to , this will assign at most one element of to every element of

Computable = when referred to a specific model of computation (usually Turing Machines, here also URM machines). Recall the definition given [here](#_Algorithms,_effective_procedures,):

* + A computable problem/function is one where the steps of computation are precisely defined and execution of the algorithm will always terminate in a finite number of steps, yielding a well-defined output.
  + Examples of computable problems include addition, multiplication, exponentiation for integer inputs, as well as problems like determining if a number is prime or solving linear equations - all of which can be solved by unambiguous, terminating algorithms.
  + In contrast, problems like the halting problem are not computable as there is no single algorithm that can correctly determine the behavior of all programs in a finite number of steps.

In computability theory, a general recursive function is a partial function from the integers to the integers; no algorithm can exist for deciding whether an arbitrary such function is in fact total.

Meaning of diagonalization and meaning with examples

* : primitive recursive function given by the step of enumeration, so it can be different from the function (so, subinputs over following computations of a function). Usually, this is useful for diagonalization arguments, which is different from

Immagine che contiene testo, calligrafia, Carattere, schermata

Descrizione generata automaticamenteImmagine che contiene testo, schermata, Carattere, calligrafia

Descrizione generata automaticamenteLet’s think on the actual proof…

The idea of these exercises is to build a function which itself is built to be different from every single other function of the same family, otherwise it is undefined.

Immagine che contiene testo, calligrafia, Carattere, numero

Descrizione generata automaticamenteThen, we have that the recursion of the function, say , which happens because the domain is built to be different from every value. An example that puts this explicitly:

* In this case, consider the function are total
  + So, they have to define and handle all cases by definition
* 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 since this holds by construction (just use the problem conditions replacing with )

Description of functions and subfunctions, which may be finite or not

* : finite subfunction
  + used in Rice-Shapiro context to show there is at least one part which has properties the rest of the considered set does not have

**Immagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamente**

* **URM machine**

It has different instructions:

* *zero* , which sets the content of register to zero:
* *successor* , which increments by 1 the content of register :
* *transfer* , which transfers the content of register into , which staying untouched:
* *conditional jump*: , which compares the content of register and , so:
  + if then jumps to (jumps to -th instruction)
  + otherwise, it will continue with the next instruction

Existence of URM-computable functions

Immagine che contiene testo, Carattere, schermata, algebra

Descrizione generata automaticamente

Model of computation: also like Turing Machines and whatnot.

Concepts of: register configuration, program counter

Introduces the concept of memory, convergence and divergence (possibly partial).

How to structure such program proofs:

* Like a TM 🡪 variant contained inside of normal (subseteq) and vice versa
* It has to be well-formed (without loss of generality – wlog)
* Has to be shown by induction on the number of steps of execution by a program
* Usually, it happens on some kind of jump to subroutines then loop
* Immagine che contiene testo, Carattere, schermata, linea

  Descrizione generata automaticamenteHas to happen that “every program executes such that ” on the same number of instructions

Let C^f be the class of functions computable by URM programs in this subclass. We will prove that C^f ⊆ C and that all functions in C^f are total.

Step 1: Show C^f ⊆ C

This is trivial as any program in our subclass is also a valid URM program.

Step 2: Prove totality

Let P be a program in this subclass. We will prove that P always terminates, thus computing a total function.

Without loss of generality (wlog), assume P is in standard form, i.e., if it terminates, it does so at instruction l(P) + 1.

We prove by induction on the number of execution steps t that after t steps, P is executing an instruction with index j > t.

Base case (t = 0):

At step 0, P is at its first instruction I\_1. This holds trivially as 1 > 0.

Inductive hypothesis:

Assume that for some k, 0 ≤ k < l(P), after k steps, P is executing instruction I\_j with j > k.

Inductive step:

We need to prove that after k+1 steps, P will be executing an instruction I\_l with l > k+1.

Case 1: If I\_j is not a jump instruction, then P moves to instruction I\_(j+1). Since j > k, we have j+1 > k+1.

Case 2: If I\_j is a jump instruction J(m,n,t), then by the condition of the subclass, t > j. Since j > k, we have t > k+1.

In both cases, for step k+1, P moves to an instruction with index greater than k+1.

By the principle of mathematical induction, for all steps t, 0 ≤ t < l(P), P is always executing an instruction with index greater than t.

Therefore, P will inevitably reach a step l(P) where it attempts to execute a non-existent instruction I\_(l(P)+1), causing termination.

Formally, we can state:

∀P ∈ subclass, ∀x ∈ N^k, ∃t ≤ l(P) such that P(x) terminates in t steps.

This proves that for any f ∈ C^f, ∀x ∈ N^k, ∃y ∈ N such that f(x) = y.

Therefore, all functions in C^f are total.

Conclusion:

We have shown that C^f ⊆ C and that every f ∈ C^f is total. Thus, the functions computable by programs in this subclass are all total.

* Immagine che contiene testo, schermata, Carattere, linea

  Descrizione generata automaticamente**Decidable predicates**

Description of meaning of characteristic function.

In mathematics, we often want to express *properties*. Consider as mathematical property the *divisor*:

Immagine che contiene testo, Carattere, bianco, calligrafia

Descrizione generata automaticamenteAs computer scientists, we can also see the divisor as a function:

In the context of computability and formal logic, we introduce the concept of a predicate, which is a statement or function that takes one or more inputs and evaluates to either *true* or *false*, typically based on some condition or relationship.

They have exercises: quote Structure Theorem and Projection Theorem (see examples)

**Immagine che contiene testo, calligrafia, Carattere, schermata

Descrizione generata automaticamenteComputability on other domains**

It’s important since we can easily define the functions by cases and “be covered” under all circumstances.

* **Generation of computable functions**

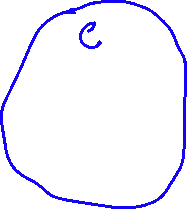
A function will be *computable* if it can be obtained from a set of basic operations that are known to be computable. Essentially, we show that having two functions we produce an operation inside in a way that composing them (for example, via ) is still in .

Immagine che contiene testo, Carattere, bianco, algebra

Descrizione generata automaticamenteWe define the composition, given you define

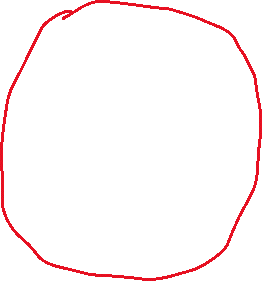
(In words: the composition function will be made on all the subfunctions if they all halt)

The class will be closed under:



* (generalized) composition
* primitive recursion
* unbounded minimalisation

To prove a function is computable, we can write a URM program of use the closure theorems of choosing the operations carefully (the ones listed above).



* **Primitive recursion**

Immagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamenteA whole set of functions do this: sum, product, exponential, predecessor, difference, sign, negative sign.

Immagine che contiene testo, calligrafia, Carattere, schermata

Descrizione generata automaticamente