Decidability and Computability

Alessandro Barenghi

Dipartimento di Elettronica e Informazione
Politecnico di Milano

alessandro.barenghi@polimi.it

April 28, 2021

What is decidability?

"Can I do this?"

- Decidability: determining if a problem can be solved in finite time
- The complexity of the computation and other resources are not taken into account (arbitrarily high, but not infinite)
- Quite important in real life: no sense in putting any effort to solve an undecidable problem
- Key point : Prove that something is decidable (or undecidable)

Decidability and semidecidability

Exactly half of decidable

- Decidability: after a finite number of steps of an algorithm, I always get a correct answer
- A weaker condition is semidecidability: after a number of steps I may get an answer in certain conditions, but the computation is not guaranteed to terminate for all inputs
- Example : I have a black box outputting numbers: will it ever output 42?
  - Semidecidable: if it outputs 42, the answer is yes; if not I cannot say anything

Problems and solutions

Set enumerability

- Given a set S, the characteristic function of S is defined as

$$\mathbf{1}_S(x) = \begin{cases} 1 & \text{if } x \in S \\ 0 & \text{if } x \notin S \end{cases} \quad (1)$$

- If $\mathbf{1}_S(x)$ is total and computable, S is *decidable* (or *recursive*)
- If $\mathbf{1}_S(x)$ is defined for the $x \in S$ case (but not necessarily for the $x \notin S$), S is *semi-decidable* (or *recursively enumerable*).
- Analogously to decidability:
  - S recursive $\Rightarrow$ S recursively enumerable
  - S recursive $\Leftrightarrow ((S \text{ r.e.}) \wedge (S^C \text{ r.e.}))$

Rice's Theorem

Does this function make coffee?

- Given a set F, of computable functions, the set of Gödel numbers of the functions in F is decidable if and only if
  - $F = \emptyset$ or
  - F is the set of all computable functions
- In other words, deciding which set of computable functions share a certain property is decidable if and only if the property is trivial (either all the functions have it or none does)

Summary

Set computability

S	S^C
computable	computable
non computable	computably enumberable
computably enumberable	non computable
non computable	non computable

- If a problem p is decidable, its specialization is decidable
- If a problem p is undecidable, its generalization is undecidable

Proofs

Proving undecidability

- "Easiest" way: if there's an algorithm that solves the problem for all possible inputs, it's decidable for sure
- If an algorithm cannot be found, we do not know anything a-priori
- More general methods can be used to prove that a problem is decidable or undecidable:
  - Reduction to a known decidable/semidecidable/undecidable problem
  - Diagonal enumeration techniques
  - Demonstrations via reductio ad absurdum
  - Rice's theorem
  - Considering the complementary problem

Example

A decidable problem

- Problem: is it possible to decide if a generic function over $\mathbb{N} \to \mathbb{N}$ has the following property:

$$\forall x((\neg(x \geq 0) \Rightarrow f(x) \neq 5) \vee$$

$$((x \geq 0) \Rightarrow (f(x) > 37 \vee (\neg(f(x) = \bot) \Rightarrow (f(x) < 100)))))$$

  - $\hookrightarrow$ $\neg(x \geq 0) \Rightarrow f(x) \neq 5$ is always true, as we are on $\mathbb{N}$
  - $\hookrightarrow$ $\neg(f(x) = \bot) \Rightarrow (f(x) < 100)$ allows to ignore cases where the function is not defined (i.e. partial)
    - In short, when $f(x) \neq \bot$, we get $\forall x(\top \vee ((x \geq 0) \Rightarrow ((f(x) > 37) \vee (f(x) < 100))))$, which is true $\forall x \in \mathbb{N}$, thus the problem is decidable, being trivially decided

Reduction

Proving undecidability

- Problem: given a generic function $f(x)$, can I establish if its domain D_f is $2\mathbb{N}$ (the set of even numbers)? (i.e. if $\forall x \in 2\mathbb{N}, f(x) \neq \perp$)
- By definition $2\mathbb{N} \subseteq \mathbb{N}$, but $|2\mathbb{N}| = |\mathbb{N}|$
- Intuition: this looks like deciding if a function over $\mathbb{N}$ is total: not decidable!
- We can prove it by reducing the problem to the aforementioned one

Reduction

Proof

- Define a function $g(y)$ as

$$g(y) = \begin{cases} f(x) & \text{if } y = 2x, x \in \mathbb{N} \\ 0 & \textit{otherwise} \end{cases}$$

- Everywhere $f(x)$ is computable, so is $g(y)$
- Deciding whether $g(y)$ is computable over $2\mathbb{N}$ implies that we can decide if $f(x)$ is computable over $\mathbb{N}$
- But deciding if a generic $f(x)$ is total over $\mathbb{N}$ is not decidable (specifically, it's undecidable), so our problem is not decidable.

Reductio ad absurdum

Proof by contradiction

- A useful tool in proving that a problem is not decidable is the proof by contradiction
- Willing to prove $\text{hypothesis}_a \Rightarrow \text{thesis}_a$:
  1. Assume thesis_a is false
  2. Deduce that $\text{thesis}_a \Rightarrow \bot$
  3. Thus thesis_a must be true
- Useful in the cases where a direct reduction to a known problem is difficult

Reductio ad absurdum

Paradoxes

- A typical reasoning to obtain a paradox is based on self referencing a concept, while inserting a negation (through logical reasoning)
- Classic example : Russell's Barber
  - A country has the following law "The barber will shave all and only the people who do not shave by themselves"
  - So, who shaves the barber?
    - He does! $\Rightarrow$ He shaves by himself $\Rightarrow$ He should not be shaven by the barber, but he's the barber Contradiction!
    - He doesn't! $\Rightarrow$ He does not shave himself $\Rightarrow$ He should be shaven by the barber, but he's the barber Contradiction!

Reductio ad absurdum

Exercise - 1

- Notation : $f_a(\cdot)$ is the function computed by the a-th Turing Machine according to Gödel enumeration of TMs.
- Given:

$$g(x,y,z) = \begin{cases} 1 & \text{if } f_z(x) = y, x \in \mathbb{N} \\ 0 & \textit{otherwise} \end{cases} \tag{2}$$

  is $g(x,y,z); x,y,z \in \mathbb{N}$ computable?
- No. Proof by contradiction :
- Assume $g(x,y,z); x,y,z \in \mathbb{N}$ is computable.

Reductio ad absurdum

Exercise - 2

- From this, we deduce that also the function

$$h(x) = g(x, 0, x) = \begin{cases} 1 & \text{if } f_x(x) = 0, x \in \mathbb{N} \\ 0 & \textit{otherwise} \end{cases} \tag{3}$$

  is computable.
- Since $h(x)$ is computable, there is a TM computing it. Call the Gödel number of that TM x_h.
- Now, try to compute $f_{x_h}(x_h)$ ...

Reductio ad absurdum

Exercise - 3

- According to (3), $f_{x_h}(x_h)$ is

$$f_{x_h}(x_h) = h(x_h) = \begin{cases} 1 & \text{if } f_{x_h}(x_h) = 0, \\ 0 & \textit{otherwise} \end{cases} \tag{4}$$

- If $f_{x_h}(x_h) = 1$, by (4) $h(x_h) = 1$, thus by (4) $f_{x_h}(x_h) = 0$
- If $f_{x_h}(x_h) = 0$, by (4) $h(x_h) = 0$, thus by (4) $f_{x_h}(x_h) \neq 0$
- In either case there is a contradiction, thus the thesis is false
- $g(x, y, z)$ is not computable

Reductio ad absurdum

A slight variant

- What if the problem of the previous exercise is modified considering

$$g(x,y,z) = \begin{cases} 1 & \text{if } f_z(x) = y, x \in \mathbb{N} \\ 0 & \textit{otherwise} \end{cases} \tag{5}$$

  is $g(x,y,z); x,y,z \in \mathbb{N}$ where z is the Gödel number of a Turing Machine computing a total function?
- This case is decidable: if $f_z(\cdot)$ is evaluable for all values of x, it is always possible to determine whether $f_z(x) = y$

Diagonal Enumerations

Bugfree™

- Is it decidable if a functional procedure (i.e. a common program) has a side-effect?
- We can reduce the halting problem to this one
- Consider a generic piece of code `P`. We can always build a function `fun` as:

```
fun(int par){
  /* Variable declarations */
  P;
  i=42; /* side effect */
};
```

- Knowing whether the side effect is run is equivalent to know if `P` terminates its execution: not decidable

Diagonal Enumerations

Bugfree™

- Is it at least semi-decidable?
- We could try to run `fun()` for all the possible values of the parameter and check whether the execution terminates properly
- Problem : what if `fun(1)` loops infinitely? We will never run `f(2)`!
- We need a smarter way to run the tests...

Diagonal Enumerations

Bugfree™

- Solution: Diagonal enumeration technique
- Consider the execution space as $\text{steps} \times \text{inputs}$
- Instead of running Step(0,`f(0)`), Step(1,`f(0)`), Step(2,`f(0)`)... (enumerate the space sweeping the first coordinate), we proceed diagonally.
- The new execution order is Step(0,`f(0)`), Step(1,`f(0)`), Step(0,`f(1)`), Step(2,f(0))...
- With this technique, if any input of `f(par)` terminates, the procedure will eventually find it
- A diagonal technique can be used for any finite number of parameters

Rice's Theorem

Exercise

- Problem: Given the set C of all functions which have constant output on their domain, is this set recursive?
- No by Rice's theorem, since :
  - $C \neq \emptyset$: for instance $f(x) = 1$ belongs to the set C
  - $C \neq \{\text{all computable functions }\}$: $f(x) = 3x$ is not in C
- Is it at least r.e.?
  - No, as it is not recursive, and its complement is r.e.
  - C^C can be enumerated using diagonal enumeration to check if there is a pair of inputs for which the output of a certain $f(x)$ is different
  - If such a pair is found, f belongs to C^C, but I cannot ever state definitely that $f \notin C^C$

Membership Problem: Diagonal Proof

Employing a proof by contradiction based on the diagonal technique, prove that, given a generic TM M with input alphabet I, the membership problem for a string $x \in I^*$ (i.e. determining if $x \in L(M)$) is not decidable.

- Consider an enumeration of strings in I^*, which is enumerable
- Define the language $L_1 = \{x_i \mid x_i \notin L(M_i)\}$
- Since the membership problem is assumed to be decidable, there exists a TM $\bar{M}$ able to recognize L_1 and $\bar{M} = M_j$ for some j
- Consider now M_j with input x_j (diagonal setting):
  - ↪ Suppose $x_j \in L(M_j)$. Then, $x_j \notin L_1 \Rightarrow$ Contradiction, since M_j recognize L_1
  - ↪ Suppose $x_j \notin L(M_j)$. Then, $x_j \in L_1 \Rightarrow$ Contradiction, since M_j recognize L_1
- We get to a contradiction in both cases, thus the problem must be non decidable

Alternative proof via reduction

We prove the previous fact by reducing the generalized halting problem

$$g(i,j) = \begin{cases} 1 & \text{if } f_i(j) \neq \perp \\ 0 & \text{if } f_i(j) = \perp \end{cases}$$

to the membership problem

- Consider a generic TM, with Gödel number i
- Consider an enumeration on I^*, and compute the string x_j corresponding to the $j-th$ element
- Then, add a looping state on the TM which is reached when the TM halts in a non-final state
- We have:
  1. $x_j \in L(M_i) \implies f_i(j) \neq \perp$ (The TM halts)
  2. $x_j \notin L(M_i) \implies f_i(j) = \perp$ (The TM does not halt)

$x_j \in L(M_i)$ is decidable $\Rightarrow$ $g(i,j)$ is computable $\Rightarrow$ Contradiction!

Problems and solutions

Quick Questions

- Is it decidable if two programs compute the same function?
  - No (undecidable), it is a generalisation of the problem of knowing whether a program computes a specific function
- Is it decidable if two programs compute the same function, knowing that both always terminate for every input?
  - No (undecidable). Even if the outputs match for all the ones we try, the possible inputs are infinite.
- Is it decidable if two programs compute the same function, knowing that both always terminate for every input and the input domain is finite?
  - Yes. It's just a matter of performing an exhaustive evaluation.

Problems and solutions

Quick Questions - 2

- Is it semi-decidable if two programs, which terminate for every input value, compute different functions?
  - Yes. Run the two programs with the same input and check if the outputs match. If the two functions are different, eventually this will be detected.
- Is it semi-decidable if two functions, with the same definition domain, are different?
  - Yes. If we know that the domains are the same, we can compute the functions for all input values such that $f_1(x) \neq \perp$ and $f_2(x) \neq \perp$ and check if the outputs match. As before, if they are different, this will be detected.
- Is it semi-decidable if two functions are different?
  - No. Two functions f_i, f_j may differ for a single value x such that $f_i(x) = \perp$ while $f_j(x) \neq \perp$. In this case, we cannot be sure that the computation on f_i will not halt with image $f_j(x)$

Buffer Overflow

Consider the problem of deciding if a generic program is immune from buffer overflow vulnerabilities. Is it decidable?

↪ No. Same reduction as side effects example! The halting problem can be reduced to this one!

Is it at least semi-decidable?

Intuition

- We can use the diagonal enumeration of computations
- But the problem is that we have to verify that every computation does not reach the vulnerable code
- We cannot be sure that non-halting computations will never reach the vulnerable code sooner or later
- The problem seems undecidable

Buffer Overflow

Proof

Consider the complement of the problem: verifying that a generic program has a buffer overflow vulnerability. Is it semi-decidable?

- In this case, diagonal enumeration works
- If there is a computation reaching the vulnerable code, we are going to find it

$$\Downarrow$$

But, since the problem is not decidable, and its complement is semi-decidable, then the problem cannot be semi-decidable

Language Classes

Brief recap

- We recall from the previous lesson that the four language classes (Type 0-3) are bound by the following relation:

$$\text{Unbounded} \subset \text{Linear Bounded} \subset \text{Context-Free} \subset \text{Regular}$$

- A couple of known facts:
  - Given two languages L_1, L_2 the problem of knowing if $L_1 \subseteq L_2$ is undecidable for both Unbounded and Context-Free
  - Knowing if a word belongs to a language is semi-decidable for Unbounded
  - Equivalence between two languages recognizable by a DPDA have been proven to be decidable

Language Classes

Exercises

- Is the inclusion problem decidable for Linear Bounded?
  - No. Linear Bounded languages are a general case of Context-Free languages, and the problem is undecidable for Context-Free.
- Is the inclusion problem decidable for Regular?
  - Regular is a special case of Context-Free, so it may be decidable. Recall that $L_1 \subseteq L_2 \Leftrightarrow L_1 \cap L_2^C \neq \emptyset$.
    Since Regular is closed w.r.t $\cap$, $L_1 \cap L_2^C$ is still in Regular.
    The emptiness problem is decidable for Regular, so the answer is Yes.

Language Classes

Exercises

- Is it decidable if the word "antani" belongs to a general Linear Bounded language?
  - Linear Bounded languages are a special case of Unbounded languages, so it might be.
  - Consider that the peculiarity of linear bounded is that, for every production rule of the grammar $\alpha \rightarrow \beta$ the following relation on the length of the sides is true : $|\alpha| \leq |\beta|$
  - This implies that a production will never decrease the string (possibly containing also nonterminals)
  - Starting from the axiom, apply all the productions allowed while keeping the string length within the one of "antani"
  - If "antani" is one of the terminal strings produced, then it belongs to the language, otherwise it cannot. The problem is decidable

Language Classes

Questions

1. If a grammar G has some production of the type $\alpha \rightarrow \beta \mid |\alpha| > |\beta|$, is the membership problem semi-decidable for $L(G)$?
   - ↪ Yes. Every grammar generates a RE language
2. If a grammar G has some production of the type $\alpha \rightarrow \beta \mid |\alpha| > |\beta|$, is the membership problem non decidable for $L(G)$?
   - ↪ No. Even a regular language can be generated by an unrestricted grammar, but the language is still decidable. However, the problem is non decidable for unrestricted languages, which are necessarily generated by an unrestricted grammar

Language Classes

Consider the following grammar G:

1. $S \to KX \mid aY$
2. $X \to aS \mid KXX$
3. $Y \to KZ \mid KS \mid aYY$
4. $K \to bZ \mid cZ$

Is decidable if $L(G) = \emptyset$?

- Yes. The problem refers to a specific grammar, therefore this is a single question with an answer Yes/No which is trivially computable, even if we may not know the right answer.
- Yes. Since some non terminal symbols cannot be erased, no string can be generated, and thus the language is empty. Hence, the problem is trivially decided and so decidable

Decidability Problems on Turing Machines

A TM M_i (computing a function f_i) is said *reproducible* if exists another M_j (computing a function f_j), such that $f_i = f_j$. Consider the set of TM defined on the input alphabet 0,1. In this set, consider these 3 subsets:

- **F** : the set of functions computed by a *reproducible* TM
- **G** : the set of functions computed by a *reproducible* TM and with less than 10 states
- **H** : the set of functions computed by a *reproducible* TM and with more than 10 states

Determine the decidability of these 3 problems:

- Decide if a generic TM computes a function in **F**
- Decide if a generic TM computes a function in **G**
- Decide if a generic TM computes a function in **H**

Decidability Problems on Turing Machines

Problem for **F**

Focus on the definition of *reproducible* TM: given a generic TM, it is always possible to build an infinite number of equivalent TMs which compute the same function!

↪ Example: add somewhere a cycle which writes n symbols on a tape, and then erase them

Therefore, each TM is *reproducible*

$$\Downarrow$$

F is the set of all computable functions!

$$\Downarrow$$

The problem is decidable because of Rice's theorem

Decidability Problems on Turing Machines

Problem for **G**

- $\mathbf{G} \neq \emptyset$: There are TM with less than 10 states computing a function
- $\mathbf{G} \neq$ the set of all computable functions: **G** is finite, while the latter is infinite

The problem of stating if a generic TM is in **G** is not decidable (Rice's theorem)

- N.B. Don't misunderstand the question as "is the set **G** computable/recursive?". Indeed, **G** is recursive!

Decidability Problems on Turing Machines

Problem for **H**

For each TM in G, we can build an equivalent TM with more than 10 states

$\hookrightarrow$ Example: add somewhere a cycle which writes 11 symbols on a tape, and then erase them

H is the set of all computable functions! $\Rightarrow$ The problem is decidable because of Rice's theorem

Decidability Problems on Turing Machines

Determine the decidability of the following problem:

$$g(i,x) = \begin{cases} 1 & \text{if, during computation of } f_i(x), \text{ there exists 2} \\ & \text{configurations of the TM } M_i \text{ with the same state} \\ 0 & \text{otherwise} \end{cases}$$

- The number of states of a TM, $|\mathbf{Q}|$, is finite
  - Therefore, if the computation requires more than $|\mathbf{Q}|$ step, there is necessarily a cycle, and thus $g(i,x) = 1$
  - If the string is recognized with less than $|\mathbf{Q}|$ steps, it is sufficient to detect a cycle
- In both cases, the output of the function can be computed with at most $|\mathbf{Q}| + 1$ steps

Decidability Problems on Turing Machines

Consider now a variant of the problem:

$$g(i,x) = \begin{cases} 1 & \text{if, during computation of } f_i(x), \text{ there exist a} \\ & \text{configuration, different from the initial one,} \\ & \text{where the TM } M_i \text{ is in the initial state} \\ 0 & \text{otherwise} \end{cases}$$

Is it decidable?

- Intuition: If the TM is looping, it may be possible that sooner or later it will stop looping and reach the initial state again
- Resembles the halting problem reasoning for its undecidability
- Let's try to reduce halting to this problem

Decidability Problems on Turing Machines

Reduction

- Pick a generic TM M
- Add a new initial state q_0', which performs only one transition to q_0 as the first step of the computation
- For each final state o M, add a transition to q_0', and mark it as the only final state
- With such a construction we obtain a TM M' which gets back to the initial state if and only f M halts

Therefore, if the problem is decidable, then the halting problem is decidable too $\Rightarrow$ Contradiction

$$\Downarrow$$

The problem is not decidable.

Determine Properties of a Function

Definition

A TM is said *n-generator* if, when its input is 0, it halts in at most $n+1$ steps, writing on the output tape n
Given an algorithmic enumeration $\mathcal{E}$ on TMs, the minimum generator of n, denoted with $gm(n)$, is the minimum index, according to $\mathcal{E}$, of an *n-generator* TM.

Is $gm(n)$ total?

- Given n, it is always possible to build an *n-generator* TM
- When the character 0 is read, write n on the output tape, which requires at most $\lceil \log_2(n) \rceil$ steps
- Since this TM always exists, we are sure that $gm(n)$ will find it even if no other *n-generator* TMs exist

$gm(n)$ is total

Determine Properties of a Function

Is $gm(n)$ computable?

- In order to verify if a TM is an *n-generator* one, it is sufficient to run at most $n+1$ steps
- Therefore, to compute $gm(n)$ is sufficient to run, with input 0, for at most $n+1$ steps each TM in increasing order according to $\mathcal{E}$
- For each TM, test if it halts after at most $n+1$ steps, writing n on the output tape
- Since $gm(n)$ is total, sooner or later an *n-generator* TM is found
- Return the index of this TM as the outcome of $gm(n)$ computation

Choosing the Right Tool From the Toolbox

Consider the function:

$$g(x,y) = \begin{cases} 1 & \text{if } \forall z(f_x(z) = f_y(z)) \\ 0 & \text{otherwise} \end{cases}$$

Is $g(x,y)$ decidable?

- Intuition: It is not, since we cannot verify that property because one of the functions may not halt for some z
- Therefore, we may try to focus on a specialization of the problem: if this is undecidable, then the generalization is undecidable too
- Which specialization?
- Consider y to be fixed at a value j, denoting a function f_j

The specialization can be formalized as:

$$g(x) = \begin{cases} 1 & \text{if } \forall z(f_x(z) = f_j(z)) \\ 0 & \text{otherwise} \end{cases}$$

Choosing the Right Tool From the Toolbox

- The problem of determining if $\forall z(f_x(z) = f_j(z))$, is equivalent to determine if a generic TM index $x \in S, S = \{j\}$.
- Since we are dealing with TM indexes, then Rice's theorem!

$$\Downarrow$$

Since this set S is neither empty (there is j), nor the universal one(there is only j), then the problem is not decidable because of Rice's theorem

Choosing the Right Tool From the Toolbox

Consider now the following function:

$$g(x, y, z) = \begin{cases} 1 & \text{if } f_x = f_y + f_z \\ 0 & \text{otherwise} \end{cases}$$

Is $g(x, y, z)$ decidable?

- Again, it seems not, as the problem resembles equivalence of a function to another, which is undecidable
- We can consider again a specialization and try to reduce the equivalence problem to this specialization

Choosing the Right Tool From the Toolbox

Reduction for $g(x, y, z)$

- Consider a fixed function f_j, and a generic one f_x
  - ↪ We know, as a consequence of Rice's theorem, that deciding if $f_x = f_j$ is undecidable.
- We can always write f_j as the sum of 2 functions f_i, f_k: trivially, pick $f_i = 0, f_k = f_j$.
- Now, consider the problem

$$g(x) = \begin{cases} 1 & \text{if } f_x = f_i + f_k \\ 0 & \text{othrwise} \end{cases}$$

- $g(x) = 1 \iff f_x = f_i + f_k = f_j$. Therefore, $g(x)$ is not decidable
- Since $g(x)$ is a specialization of $g(x, y, z)$, then $g(x, y, z)$ is undecidable too.

A Slight Variant

Consider the function

$$g(x) = \begin{cases} 1 & \text{if } f_x = f_i + f_k \wedge \forall z(f_x(z) = h) \\ 0 & \text{otherwise} \end{cases}$$

Where i, k and h are fixed values. Is $g(x)$ decidable?

- We have 2 cases.
  1. $f_i + f_k$ is not a constant function.
  2. $f_i + f_k$ is a constant function.

Case 1

The problem is trivially decided: the answer is always 0, since f_x cannot be both constant and equal to $f_i + f_k$

A Slight Variant

Case 2

The problem becomes

$$g(x) = \begin{cases} 1 & \text{if } \forall z(f_i(z) + f_k(z) = f_x(z) = h) \\ 0 & \text{otherwise} \end{cases}$$

- We can see $f_i + f_k$ as the function $f_j(x) = h \, \forall x \in \mathbb{N}$
- The problem is thus equivalent to determine if $f_x = f_j$, which is non decidable as a consequence of Rice's theorem

Functions with Even Images

Consider these 4 similar problems:

1. $$g(y,x) = \begin{cases} 1 & \text{if } f_y(x) \text{ is even} \\ 0 & \text{otherwise} \end{cases}$$
2. $$g(y,x) = \begin{cases} 1 & \text{if } f_y(x) \text{ is even} \\ \perp & \text{otherwise} \end{cases}$$
3. $$g(y) = \begin{cases} 1 & \text{if } f_y(x) \text{ is even } \forall x \in \mathbb{N} \\ 0 & \text{otherwise} \end{cases}$$
4. $$g(y) = \begin{cases} 1 & \text{if } f_y(x) \text{ is even } \forall x \in \mathbb{N} \\ \perp & \text{otherwise} \end{cases}$$

Functions with Even Images

Problem 1

We can reduce the halting problem to this one. Indeed, consider a generic function f_y. Define:

$$h(y,x) = \begin{cases} 1 & \text{if } 2f_y(x) \text{ is even} \\ 0 & \text{otherwise} \end{cases}$$

Deciding if $h(y,x)$ is even, allows to decide if $f_y(x)$ halts $\Rightarrow$ Contradiction!

$$\Downarrow$$

Problem 1 cannot be decidable, otherwise the halting problem would be decidable

Functions with Even Images

Problem 2

Simply run $f_y(x)$ and look at the result:

- If $f_y(x) \neq \perp \wedge f_y(x)$ is even, return 1
- If $f_y(x) \neq \perp \wedge f_y(x)$ is odd, move the TM in a looping state
- If $f_y(x) = \perp$, the TM is not halting as required by the function

Problem 3

Consider the set S of Gödel numbers of TM which always compute even values

- $S \neq \emptyset$: $f(x) = 2x \in S$
- $S \neq$ the set of all computable functions: for instance, $f(x) = x \notin S$

Since $g(y)$ is the characteristic function of S, the problem is not decidable because of Rice's theorem

Functions with Even Images

Problem 4

Deciding this problem is equivalent to a positive answer of problem 3

$$\Downarrow$$

Equivalent to establish if Problem 3 is semi-decidable!

- Consider a generic function $f_y(x)$
- Define:

$$h(y) = \begin{cases} 1 & \text{if } 2f_y(x) \text{ is even } \forall x \in \mathbb{N} \\ \bot & \text{otherwise} \end{cases}$$

- Decide if $h(y) = 1$ entails deciding if $f_y(x)$ is total
- Since the set of total functions is not RE, then $h(y)$ cannot be decided too, and thus Problem 4 is not decidable