# Rice's Theorem

## A strong and general undecidability result

Scuola Ortogonale, II ciclo

**Samuele Manclossi** (Università degli Studi di Milano)

Mentor: Prof. Paolo Boldi

Third Orthogonal Weekend, Bertinoro 20-22 March

elicsir
Elevating Italian Computer Science Innovation and Research

The development of a strong mathematical characterisation of the class of computable functions has been instrumental for modern computer science, yet it made also possible to discover some darker results.

# Table of Contents

# History



*Given a Diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: To devise a process according to which it can be determined in a finite number of operations whether the equation is solvable in rational integers.*

Hilbert's tenth problem

# History



- A unique definition of a process didn't properly exist at the time
- Different mathematicians and logicians started working on it
- Their purpose was to give a mathematical definition of what is computable

# History



- After several years of research, such idea was found
- Recursion theory was established by Turing, Post, Kleene, Péter, Gödel and Church
- Progressively, some discouraging results began to appear...

# A mathematical idea of computation

Dedekind 1888, Skolem 1923, Gödel 1931, Kleene 1936

- The mathematical idea of computation defines a class of intuitively computable functions
- Intuitively, we can compute the following initial functions:
  - $\mathcal{O}(x) = 0$ (the constant zero function)
  - $\mathcal{S}(x) = x + 1$ (the successor function)
  - $\mathcal{I}_i^n(x_1, \ldots, x_n) = x_i$ (the identity or projector function)

# A mathematical idea of computation

Dedekind 1888, Skolem 1923, Gödel 1931, Kleene 1936

- There are also some operations for which we want this class of functions to be closed:
  - composition: if $g_i, h$ are computable then also $f$ is, where $f$ is defined as

$$f(x_1, ..., x_n) = h(g_1(x_1, ..., x_n), ..., g_m(x_1, ..., x_n))$$

  - primitive recursion: if $g, h$ are computable then also $f$ is, where $f$ is defined as

$$f(x_1, ..., x_n, y) = \begin{cases} g(x_1, ..., x_n) & \text{if } y = 0 \\ h(x_1, ..., x_n, y-1, f(x_1, ..., x_n, y-1)) & \text{otherwise} \end{cases}$$

# A mathematical idea of computation

Dedekind 1888, Skolem 1923, Gödel 1931, Kleene 1936

- The smallest class of function for which this is true is that of primitive recursive (PR) functions
- There is still an operator for which we want the class of function to be closed:
  - $\mu$-recursion or minimization: the minimal $y$ for which a predicate $R$ is true

$$\mu y R(x_1, ..., x_n, y) \stackrel{\text{def}}{=} y : R(x_1, ..., x_n, y) \wedge \forall z < y \neg R(x_1, ..., x_n, z)$$

In our context, it can be seen as:

$$\mu y[g(x_1, ..., x_n, y) = 0] \stackrel{\text{def}}{=} y : g(x_1, ..., x_n, y) = 0 \wedge \forall z < y, g(x_1, ..., x_n, y) \neq 0$$

# A mathematical idea of computation

Dedekind 1888, Skolem 1923, Gödel 1931, Kleene 1936

- The smallest class of functions now identified is that of partial recursive functions
- The $\mu$-operator leads to a formulation which cannot always be defined, for example $R(y \in \mathbb{N}) : g(y) = 0$, where $g(y) = y + 1$
- A partial recursive function $\varphi$ can be total if it is defined for all inputs.

# Gödel numbering and countability

Gödel 1931

- It is possible to write each partial recursive function through a finite syntactic description
- Hence, it can be encoded using Gödel's numbering (sometimes called arithmetization)
- Therefore the class of partial recursive functions is countable and can be expressed as $\mathcal{P} = \{\varphi_i\}_{i \in \mathbb{N}}$ (first Rogers' axiom)

# Existence of an universal partial function

Post 1922, Turing 1936, Kleene 1938

**Theorem (Universal partial function, or second Rogers' axiom)**

*There is a partial recursive functions $\varphi_u$ s.t. $\forall \varphi_e \in \mathcal{P}, \forall x, \varphi_u(e, x) = \varphi_e(x)$.*

# Existence of an universal partial function

Post 1922, Turing 1936, Kleene 1938

The proof can be performed constructively by defining it:

- for initial functions, where $e_f$ is the index of function $f$:

$$\varphi_u(e_{\mathcal{O}}, x) = 0 \quad \varphi_u(e_{\mathcal{S}}, x) = x + 1 \quad \varphi_u(e_{\mathcal{I}_n^i}, x_1, ..., x_n) = x_i$$

- for composition, given $f(x) = h(g_1(x), ..., g_m(x))$ and $e_h, e_{g_i}$:

$$\varphi_u(e_f, x) = \varphi_u(e_h, \langle \varphi_u(e_{g_1}, x), ..., \varphi_u(e_{g_m}, x) \rangle)$$

# Existence of an universal partial function

Post 1922, Turing 1936, Kleene 1938

- for primitive recursion:

$$\varphi_u(e_f, x, y) = \begin{cases} \varphi_u(e_g, x) & \text{if } y = 0 \\ \varphi_u(e_h, \langle x, y-1, \varphi_u(e_f, x, y-1)\rangle) & \text{otherwise} \end{cases}$$

- for minimization:

$$\varphi_u(e_f, x) = \mu y[\varphi_u(e_g, x, y) = 0]$$

It uses only the definition of partial recursive and encodings, which are partial recursive, so the universal partial function is partial recursive, and identified by Gödel number $u$.

# $S_n^m$ theorem

Kleene 1943, Rogers 1987, Soare 1987

**Theorem ($S_n^m$ theorem, or third Rogers' axiom)**

$\forall m, n \geq 1\ \exists S_n^m : \mathbb{N}^{m+1} \to \mathbb{N}$ *total recursive s.t.* $\forall e, y_1, \ldots, y_m, x_1, \ldots, x_n$*:*

$$\varphi_{S_n^m(e,y_1,\ldots,y_m)}(x_1, \ldots, x_n) = \varphi_e(y_1, \ldots, y_m, x_1, \ldots, x_n)$$

It is a specialisation theorem: if a function is computable on a given input, also the function with some of these inputs hard coded is computable on the remaining. This substitution process is total recursive and is expressed by $S_n^m$.

# Rogers' axioms



The last three theorems are sometimes called Rogers' axioms, and will be used to prove the next result. They are:

1. existence of a Gödel numbering of partial recursive functions: $\mathcal{P} = \{\varphi_i\}_{i \in \mathbb{N}}$
2. existence of the universal partial recursive function: $\exists u : \varphi_u(e, x) = \varphi_e(x) \forall e, x$
3. existence of the $S_n^m$ total function:

$$\exists S_n^m : \varphi_{S_n^m(e, y_1, \ldots, y_m)}(x_1, \ldots, x_n) = \varphi_e(y_1, \ldots, y_m, x_1, \ldots, x_n) \forall e, y_i, x_i$$

# Rogers' fixed-point theorem



**Theorem (Fixed-point theorem)**

*For all* $t : \mathbb{N} \to \mathbb{N}$ *total recursive*

$$\exists n \in \mathbb{N} | \varphi_n = \varphi_{t(n)}$$

# Rogers' fixed-point theorem



We can prove that:

- $\varphi_{\varphi_i(i)}(x) \overset{(2)}{=} \varphi_{\varphi_u(i,i)}(x) \overset{(2)}{=} \varphi_u(x, \varphi_u(i,i))$ which is a recursive function in $x, i$
- Therefore exists an index $e$ s.t. $\varphi_u(x, \varphi_u(i,i)) \overset{(1)}{=} \varphi_e(i,x) \overset{(3)}{=} \varphi_{S_1^1(e,i)}(x)$ (lemma $A$)
- Let's consider $t(S_1^1(e,i))$: it is total recursive in $i$ because it is a composition of total recursive functions $\Rightarrow \exists m \in \mathbb{N} : \varphi_m(i) = t(S_1^1(e,i))$ (lemma $B$)

# Rogers' fixed-point theorem



Lemmas:

*A.* $\varphi_{\varphi_i(i)}(x) = \varphi_{S_1^1(e,i)}(x)$

*B.* $\varphi_m(i) = t(S_1^1(e,i))$

Now let's consider $n = S_1^1(e,m)$:

$$\varphi_n(x) \overset{\text{def}}{=} \varphi_{S_1^1(e,m)}(x) \overset{(A)}{=} \varphi_{\varphi_m(m)}(x) \overset{(B)}{=} \varphi_{t(S_1^1(e,m))}(x) \overset{\text{def}}{=} \varphi_{t(n)}(x)$$

# Table of Contents

2 Rice's Theorem

# Rice's Theorem

Rice 1953

**Theorem (Rice)**

*A class of partial recursive functions $\mathcal{A}$ is recursive if and only if it is trivial.*

Trivial means that it is either empty ($\mathcal{A} = \varnothing$) or it contains all partial recursive functions ($\mathcal{A} = \{\varphi_i\}_{i \in \mathbb{N}}$).

**Theorem (Rice, reformulated)**

*Any nontrivial extensional index class is undecidable.*

# (Simplest) proof



- Let $I \subseteq \mathbb{N}$ be the set of indexes of the instances which belong to $\mathcal{A}$
- Due to extensionality $(a \in I \land \varphi_a = \varphi_b) \Rightarrow b \in I$ ($I$ is semantically closed)
- The two formulations are equivalent
- Let's suppose $I \neq \varnothing$, $I \neq \mathbb{N}$ and $I$ recursive
- then $\exists a \in I, \bar{a} \notin I$

# (Simplest) proof



- Let's consider

$$t(n) = \begin{cases} \bar{a} & \text{if } n \in I \\ a & \text{if } n \notin I \end{cases}$$

- $I$ is recursive $\Rightarrow$ set membership in $I$ is decidable $\Rightarrow$ $t$ is (total) recursive
- Due to Rogers' fixed point theorem, $\forall t$ total recursive $\exists d \in \mathbb{N} : \varphi_d = \varphi_{t(d)}$
- Either $d \in I$ or $d \notin I$:
  - $d \in I$. But $\varphi_d = \varphi_{t(d)} \Rightarrow t(d \in i) \in I$ but $t(d \in I) = \bar{a} \notin I$ absurd
  - $d \notin I \Rightarrow t(d) = a \in d$ but $\varphi_{t(d)} = \varphi_d \Rightarrow d \in I$ absurd
- There cannot be such index set $\Rightarrow$ our equivalent version of Rice's theorem is true $\Rightarrow$ we proved Rice's theorem.

# Rice's Theorem

Rice 1953

Another way to state it is:

**Theorem (Rice)**

*Any nontrivial property of partial recursive functions is undecidable.*

# Rice's Theorem

Rice 1953

Consequence:

**Corollary**

*Any nontrivial semantic property of programs is undecidable.*

There can be no fully automatic* formal verification.

# Hope beyond Rice's Theorem



Maybe there are still ways to work around it:

- it could be specific to a certain computational model

# Hope beyond Rice's Theorem



Maybe there are still ways to work around it:

- it could be specific to a certain computational model
- we can survive without decidability

# Table of Contents

# Is the result general?

3 Church's thesis: generality of results

Rice theorem could be less daunting than what it seems:

- it has been proven for partial recursive functions
- what if they don't capture the true essence of computability?

# Is the result general?

3 Church's thesis: generality of results

Rice theorem could be less daunting than what it seems:

- it has been proven for partial recursive functions
- what if they don't capture the true essence of computability?
- we might simply change the computation model and ignore it!

# Is the result general?

3 Church's thesis: generality of results

Rice theorem could be less daunting than what it seems:

- it has been proven for partial recursive functions
- what if they don't capture the true essence of computability?
- we might simply change the computation model and ignore it!
- can we?

# A general approach

## 3 Church's thesis: generality of results

A general approach to find it would consist of the following two phases:

1. create a new computation model that is not equivalent to partial recursive functions
2. prove that there Rice theorem doesn't hold

# A general approach

## 3 Church's thesis: generality of results

A general approach to find it would consist of the following two phases:

1. create a new computation model that is not equivalent to partial recursive functions
2. prove that there Rice theorem doesn't hold

We will see that point 1 is the true problem.

# Flowchart computability

Goldstine and Von Neumann 1947

Programs can be seen as a combination of:

- performing simple actions
- asking simple questions

# Flowchart computability

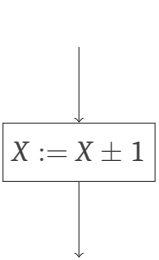
Goldstine and Von Neumann 1947

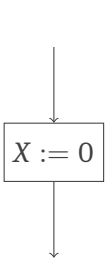
Programs can be seen as a combination of:

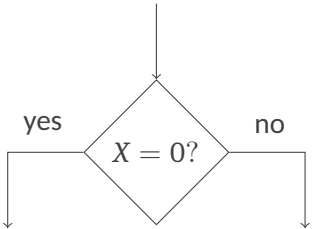
- performing simple actions $\longrightarrow$ boxes
- asking simple questions $\longrightarrow$ diamonds

# Flowchart computability

Goldstine and Von Neumann 1947



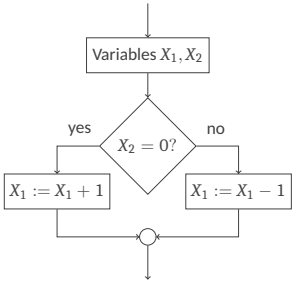



# Flowchart computability

Goldstine and Von Neumann 1947

**Definition (Flowchart computability)**

A function $f(x_1, \ldots, x_n)$ is flowchart computable if there is a flowchart program with an output variable $Z$ and s.t. if $X_1, \ldots, X_n = x_1, \ldots x_n, X_{n'>n} = 0$, at the exit point $Z = f(x_1, \ldots, x_n)$.




$$f(x_1) = x_1 + 1$$

# Flowchart computability

Goldstine and Von Neumann 1947

**Theorem (Wang 1957, Peter 1958, Ershov 1960)**

*The class of flowchart computable functions is the class of partial recursive functions.*

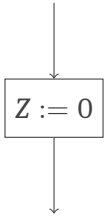
The proof by induction is trivial.

# Flowchart computability

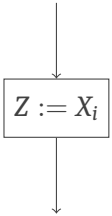
Goldstine and Von Neumann 1947

Initial functions:

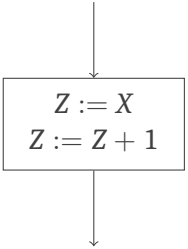
$\mathcal{O}$




$\mathcal{I}_i^n$




$\mathcal{S}$

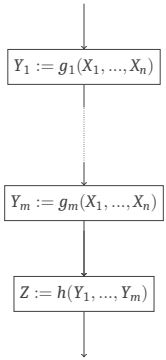


# Flowchart computability

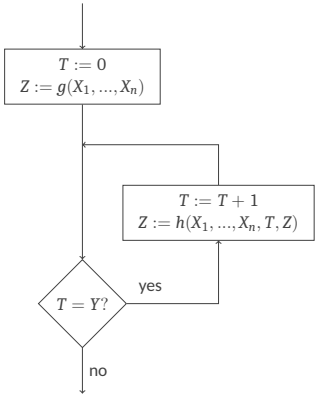
Goldstine and Von Neumann 1947

Operations:

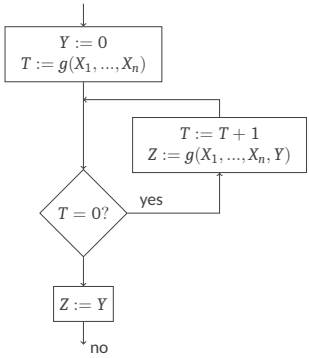
composition




primitive recursion




$\mu$-recursion



# Flowchart computability

Goldstine and Von Neumann 1947

To prove that nothing else is flowchart-computable, an intuitive proof can procede along the following steps:

- the basic actions are initial functions or recursive checks $\longrightarrow$ they are recursive
- the possible flows can be expressed as operations on partial recursive functions:
  - sequences $\longrightarrow$ composition
  - bounded loops $\longrightarrow$ primitive recursion
  - unbounded loops (while) $\longrightarrow$ $\mu$-recursion

Thus, the theorem is proven. □

# Flowchart computability

Goldstine and Von Neumann 1947

→ flowchart computability identifies the very same class of functions!

→ Rice theorem applies also here.

# Register Machine

Melzak 1961, Lambek 1961, Shepherdson and Sturgis 1963, Peter 1963, Elgot and Robinson 1964

A machine alternative to Turing's one:

- infinite registers $L, R_0, R_1, ..., R_j, ... \in \mathbb{N}$ ($L$ is a program counter)
- initially $L = 1$, $R_1$ is the input $x$, $R_{i \neq 1} = 0$
- a program is a list of register machine instructions
- execution happens by performing the $L$-th instruction and updating $L$
- when instructions end ($L > |P| \vee L = 0$) $R_0$ contains the output value

# Register Machine

Melzak 1961, Lambek 1961, Shepherdson and Sturgis 1963, Peter 1963, Elgot and Robinson 1964

Instructions can be:

- assignment:
  - $R_k \leftarrow R_k + 1$
  - $R_k \leftarrow R_k \dot{-} 1$ (natural predecessor)
- jump: *IF* $R_k = 0$ *THEN GOTO* $n$

# Register Machine

Melzak 1961, Lambek 1961, Shepherdson and Sturgis 1963, Peter 1963, Elgot and Robinson 1964

**Definition (Register Machine Computability)**

A function $f(x_1, ..., x_n)$ is register machine computable if there is a register machine program such that $\delta^*(in(< x_1, ..., x_n >)(L) = 0$ (the execution of the program terminates on these inputs) and $\delta^*(in(< x_1, ..., x_n >)(R_0) = z$, where $z = f(x_1, ..., x_n)$.

1. $R_1 \leftarrow R_1 + 1$
2. IF $R_1 = 0$ THEN GOTO 0
3. $R_1 \leftarrow R_1 \dot{-} 1$
4. $R_0 \leftarrow R_0 + 1$
5. IF $R_3 = 0$ THEN GOTO 2
6. $R_8 \leftarrow R_8 + 1$

$f(x_1) = x_1 + 1$

# Register Machine

Melzak 1961, Lambek 1961, Shepherdson and Sturgis 1963, Peter 1963, Elgot and Robinson 1964

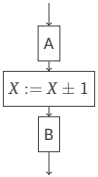
**Theorem**

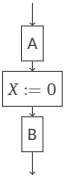
*The class of register machine computable functions is the class of partial recursive functions.*

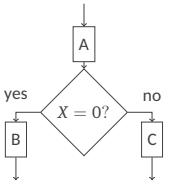
An intuitive proof can consist in showing a translation between register machine and flowcharts.

# Register Machine

Melzak 1961, Lambek 1961, Shepherdson and Sturgis 1963, Peter 1963, Elgot and Robinson 1964




```
... A ...
i+1. R_1 ← R_1 ∸ 1
i+2. IF R_1 = 0 THEN GOTO i+6
i+3. R_1 ← R_1 + 1
i+4. R_0 ← R_0 + 1
i+5. IF R_3 = 0 THEN GOTO i+2
i+6. ... B ...
```

```
... A ...
i+1. IF R_1 = 0 GOTO i+3
i+2. R_1 ← R_1 ∸ 1
i+3. ... B ...
```

```
... A ...
i+1. IF R_1 = 0 GOTO j+1
i+2. ... C ...
j+1. IF R_42 = 0 GOTO k+1
j+2. ... B ...
k+1. ...
```

# Register Machine

Melzak 1961, Lambek 1961, Shepherdson and Sturgis 1963, Peter 1963, Elgot and Robinson 1964

→ Register machines compute the same class of functions!

→ Rice theorem applies also here.

# Which other models?

3 Church's thesis: generality of results

- solving systems of equations through specific rules $\longrightarrow$ equivalent!
- using reduction rules in untyped $\lambda$-calculus $\longrightarrow$ equivalent!
- Turing's machines $\longrightarrow$ equivalent!
- While machines / IMP language $\longrightarrow$ equivalent!
- DNA and/or chemical computing $\longrightarrow$ equivalent!
- Post canonical/production systems $\longrightarrow$ equivalent!
- Type 0 formal grammars $\longrightarrow$ equivalent!
- Tactics language in Rocq/Coq $\longrightarrow$ equivalent!
- Game of life, Rule 110, Turing-complete cellular automatas $\longrightarrow$ equivalent!
- C printf format string, x86 MOV instruction, Magic: The Gathering, Doom, ...

## *A kind of miracle*

Gödel 1946

*Tarski has stressed in his lecture (and I think justly) the great importance of the concept of general recursiveness (or Turing's computability). It seems to me that this importance is largely due to the fact that with this concept one has for the first time succeeded in giving an absolute definition of an interesting epistemological notion, i.e., one not depending on the formalism chosen.*

Kurt Gödel
Remarks at the Princeton Bicentennial Conference on Problems of Mathematics
1946

# Church's Thesis

Church 1936, Turing 1936

**Thesis (Church's Thesis)**

*Every effectively computable function is recursive*

*[it would be a result] second only to that of the formulation of the concept of natural number*

Emil Post, 1944

# Church's Thesis

Church 1936, Turing 1936

- Should we believe in Church's thesis?

# Church's Thesis

Church 1936, Turing 1936

- Should we believe in Church's thesis?
- We tried not to, but we have been unable to find a counterexample.

# Church's Thesis

Church 1936, Turing 1936

- Should we believe in Church's thesis?
- We tried not to, but we have been unable to find a counterexample.
- This class seems to be extremely stable: *a bound for the potentiality of pure formalism* (Gödel).

# Church's Thesis

Church 1936, Turing 1936

- Should we believe in Church's thesis?
- We tried not to, but we have been unable to find a counterexample.
- This class seems to be extremely stable: *a bound for the potentiality of pure formalism* (Gödel).
- It seems that we can't avoid the consequences of Rice's theorem… What to do now?

# Table of Contents

# Is formal verification dead?

## 4 Beyond Rice and Church

- Absolutely not!
- How does it work around Rice's theorem?

# Re-reading the corollary

4 Beyond Rice and Church

**Corollary**

*Any nontrivial semantic property of programs is undecidable.*

Equivalent to state that we cannot have a program s.t.

- it is general: it applies to all programs
- non-triviality: some programs, but not all of them, satisfy a property
- it analyses semantic properties
- it automatically decides the decision problem: it always gives a correct yes/no answer

# Re-reading the corollary

## 4 Beyond Rice and Church

- These properties are all we could desire

# Re-reading the corollary

## 4 Beyond Rice and Church

- These properties are all we could desire
- What if we renounce to some of them?

# Deductive verification

## 4 Beyond Rice and Church

- Also known as theorem proving
- It gives up on (full) automation:
  - A human is involved providing loop invariants or proof hints
  - The process is no longer an algorithm that must decide the properties
  - It only checks if the proofs are correct
- Notable examples are tools like Rocq (ex Coq) and Lean.

# Model checking

4 Beyond Rice and Church

- It works on finite-state systems, thus giving up on generality
- Often the systems we use can be described as finite states
- This limits us to non-Turing-complete, decidable systems
- A notable example is TLA+ model checker

# Abstract interpretation (static analysis)

## 4 Beyond Rice and Church

- It gives up on completeness: does not aim to decide the exact semantic property (solving the decision problem)
- It computes an over-approximation of the program behaviour that is sound but not complete:
  - if it reports no errors, errors cannot occur during execution;
  - if it reports any error, there may or may not be errors.

# Conclusions

## 4 Beyond Rice and Church

During this presentation we have seen that

- Rice's theorem is a strong undecidability result;
- Church's thesis confirms its generality;
- there are still ways to work around it, but they imply to renounce to useful properties.

# Rice's Theorem

*Thank you for listening!*
*Any questions?*