Waiting for Gödel

Tristan Stérin
Hamilton Institute & Computer Science Department
Maynooth University, Ireland

ERC No 772766, SFI 18/ERCS/5746

An old question

$$\sum_{k=1}^{n} k = \frac{1}{2} n(n+1)$$

$$\int_{-\infty}^{\infty} e^{-x^2} dx = \sqrt{\pi}$$

$$\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6}$$

$$\frac{\pi}{2} = \prod_{n=1}^{\infty} \frac{4n^2}{4n^2 - 1} = \prod_{n=1}^{\infty} \left(\frac{2n}{2n-1} \cdot \frac{2n}{2n+1} \right)$$
$$= \left(\frac{2}{1} \cdot \frac{2}{3}\right) \cdot \left(\frac{4}{3} \cdot \frac{4}{5}\right) \cdot \left(\frac{6}{5} \cdot \frac{6}{7}\right) \cdot \left(\frac{8}{7} \cdot \frac{8}{9}\right) \cdot \cdots$$

What is the result of this computation?

A new question

What does it mean to compute?

Naïvely: Doing something in an organised/programmed way.

A new question

What does it mean to compute?

Naïvely: Doing something in an organised/programmed way.

With this question, formalised in the 30's, computer science was born!

- ~1930: Alonzo Church, Lambda Calculus
- 1936: Stephen Cole Kleene, general recursive functions
- 1936: Alan Turing, **Turing machines**
- 1943: Emil Post, Tag systems
- 1945: von Neumann, RAM model

A new question

What does it mean to compute?

These models look different:

$$
\begin{array}{lll}
 & (\lambda f.\lambda g.\lambda h.fg(h\,h))(\lambda x.\lambda y.x)h(\lambda x.x\,x) & \\
\to_\beta & (\lambda g.\lambda h.(\lambda x.\lambda y.x)g(h\,h))h(\lambda x.x\,x) & (1)\\
\to_\alpha & (\lambda g.\lambda k.(\lambda x.\lambda y.x)g(k\,k))h(\lambda x.x\,x) & (2)\\
\to_\beta & (\lambda k.(\lambda x.\lambda y.x)h(k\,k))(\lambda x.x\,x) & (3)\\
\to_\beta & (\lambda x.\lambda y.x)h((\lambda x.x\,x)\,(\lambda x.x\,x)) & (4)\\
\to_\beta & (\lambda y.h)((\lambda x.x\,x)\,(\lambda x.x\,x)) & (5)\\
\to_\beta & h & (6)
\end{array}
$$

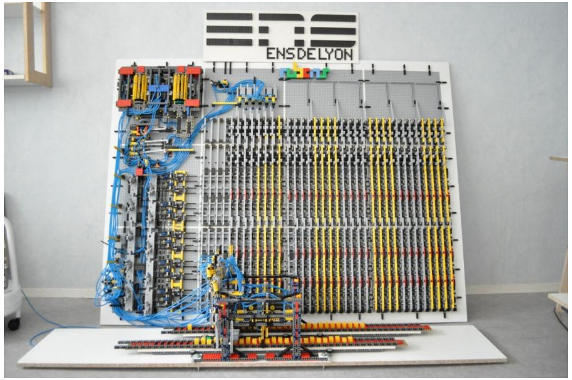
Lambda Calculus

$$
\begin{cases}
\xi(0, b, a) = a + b, \\
\xi(n', 0, a) = \alpha(n, a), \\
\xi(n', b', a) = \xi(n, \xi(n', b, a), a).
\end{cases}
$$

Recursive Functions

$$
\begin{array}{cccccccc}
g_{10} & \$_{11} & g_{11} & \$_{12} & g_{12} & \cdots & \$_{1m_1} & g_{1m_1} \\
g_{20} & \$_{21} & g_{21} & \$_{22} & g_{22} & \cdots & \$_{2m_2} & g_{2m_2} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
g_{k0} & \$_{k1} & g_{k1} & \$_{k2} & g_{k2} & \cdots & \$_{km_k} & g_{km_k} \\
 & & & & \downarrow & & & \\
h_0 & \$'_1 & h_1 & \$'_2 & h_2 & \cdots & \$'_n & h_n
\end{array}
$$

Tag Systems



Turing Machines

A new question

What does it mean to compute?

1. These models look different but they all can **simulate** one another

A new question

What does it mean to compute?

1. These models look different but they all can **simulate** one another

2. Any computation we ever thought of, we have been able to implement with a Turing machine (or any other of these models)

Church-Turing (philosophical) thesis.

Something is physically computable if and only if it can be computed by a **Turing machine**.

A low-level programming language running on an ideal primitive computer.

A new question

What does it mean to compute?

1. These models look different but they all can **simulate** one another
2. Any computation we ever thought of, we have been able to implement with a Turing machine (or any other of these models)

Church-Turing (philosophical) thesis.

Something is physically computable if and only if it can be computed by a **Turing machine**.

Naively: Doing something in an organised/programmed way.

Another question

Can we know everything?

At least, can we know everything about the natural numbers?

$$\mathbb{N} = \{0, 1, 2, \dots\} \text{ with } \{+, \times, <\}$$

At least, can we know everything about the natural numbers?

$$\mathbb{N} = \{0, 1, 2, \ldots\} \text{ with } \{+, \times, <\}$$

- The sum of two even numbers is even: **True**

At least, can we know everything about the natural numbers?

$$\mathbb{N} = \{0, 1, 2, \ldots\} \text{ with } \{+, \times, <\}$$

- The sum of two even numbers is even: **True**
- There are finitely many primes: **False**

At least, can we know everything about the natural numbers?

$$\mathbb{N} = \{0, 1, 2, \dots\} \text{ with } \{+, \times, <\}$$

- The sum of two even numbers is even: **True**
- There are finitely many primes: **False**
- If a^2 ends in the pattern xyxyxyxyxy then xy is either 21, 61 or 84: 508853989^2 = 258932382121212121. **True**

At least, can we know everything about the natural numbers?

$$\mathbb{N} = \{0, 1, 2, \ldots\} \text{ with } \{+, \times, <\}$$

- The sum of two even numbers is even: **True**
- There are finitely many primes: **False**
- If a^2 ends in the pattern xyxyxyxyxy then xy is either 21, 61 or 84: 508853989^2 = 258932382121212121. **True**
- Every integer greater than 5 can be written as the sum of 3 primes. **??** **Goldbach's conjecture.**

At least, can we know everything about the natural numbers?

$$\mathbb{N} = \{0, 1, 2, \dots\} \text{ with } \{+, \times, <\}$$

No, we can't :(

First Incompleteness Theorem (Kurt Gödel, 1931)

*For any **consistent** and **computable** set of axioms expressed in the language of arithmetic,*
There exists a statement that is true in the natural numbers but that cannot be proved from this set of axioms.

At least, can we know everything about the natural numbers?

$$\mathbb{N} = \{0, 1, 2, \ldots\} \text{ with } \{+, \times, <\}$$

No, we can't :(

First Incompleteness Theorem (Kurt Gödel, 1931)

*For any **consistent** and **computable** set of axioms expressed in the language of arithmetic,*
There exists a statement that is true in the natural numbers but that cannot be proved from this set of axioms.

Such statement is said to be "**undecidable**" with respect to the system of axioms that was chosen:

- Maybe Goldbach's conjecture is undecidable **with respect to** Peano Axiom's?
- Maybe Goldbach's conjecture is undecidable **with respect to** ZFC Axioms?

But in any case, Goldbach's conjecture is either true or false in the natural numbers.

At least, can we know everything about the natural numbers?

$$\mathbb{N} = \{0, 1, 2, \ldots\} \text{ with } \{+, \times, <\}$$

No, we can’t :(

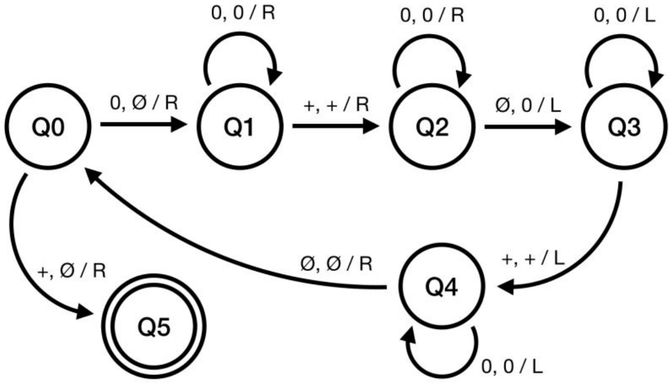
First Incompleteness Theorem (Kurt Gödel, 1931)

*For any **consistent** and **computable** set of axioms expressed in the language of arithmetic,*
There exists a statement that is true in the natural numbers but that cannot be proved from this set of axioms.

But how do we know that the statement is true if we cannot prove it??

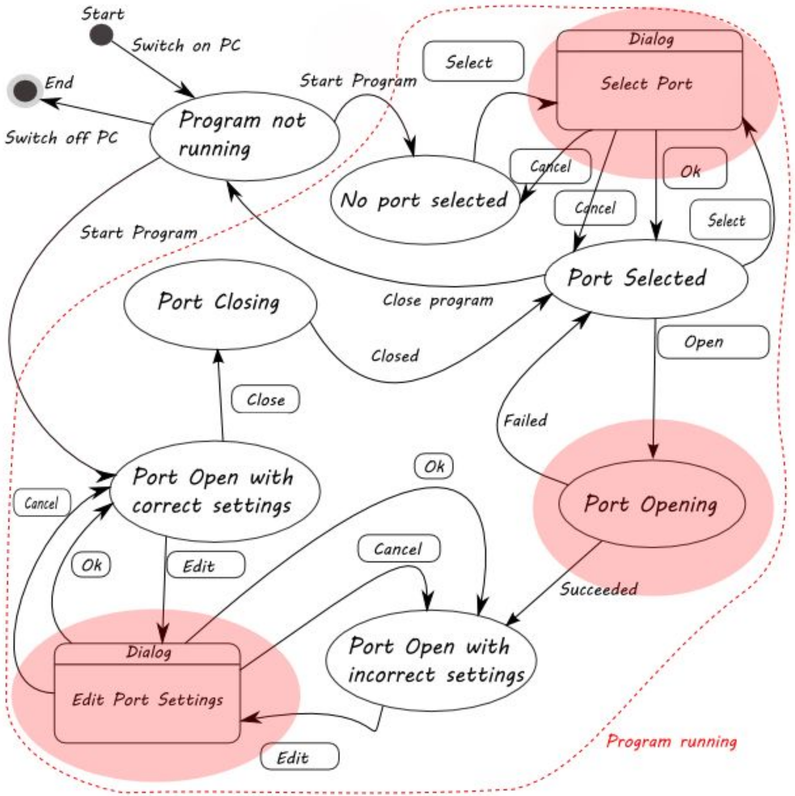
The link with **Turing Machines** will make this clear.

Turing Machines



Traditionally represented like above but, arguably, we loose all programmatic intuition with this representation!

Turing Machines



"This is a Python program"

Turing Machines

A Turing machine is a primitive (ideal) **computer architecture** together with a primitive **programming language**.

Turing Machines

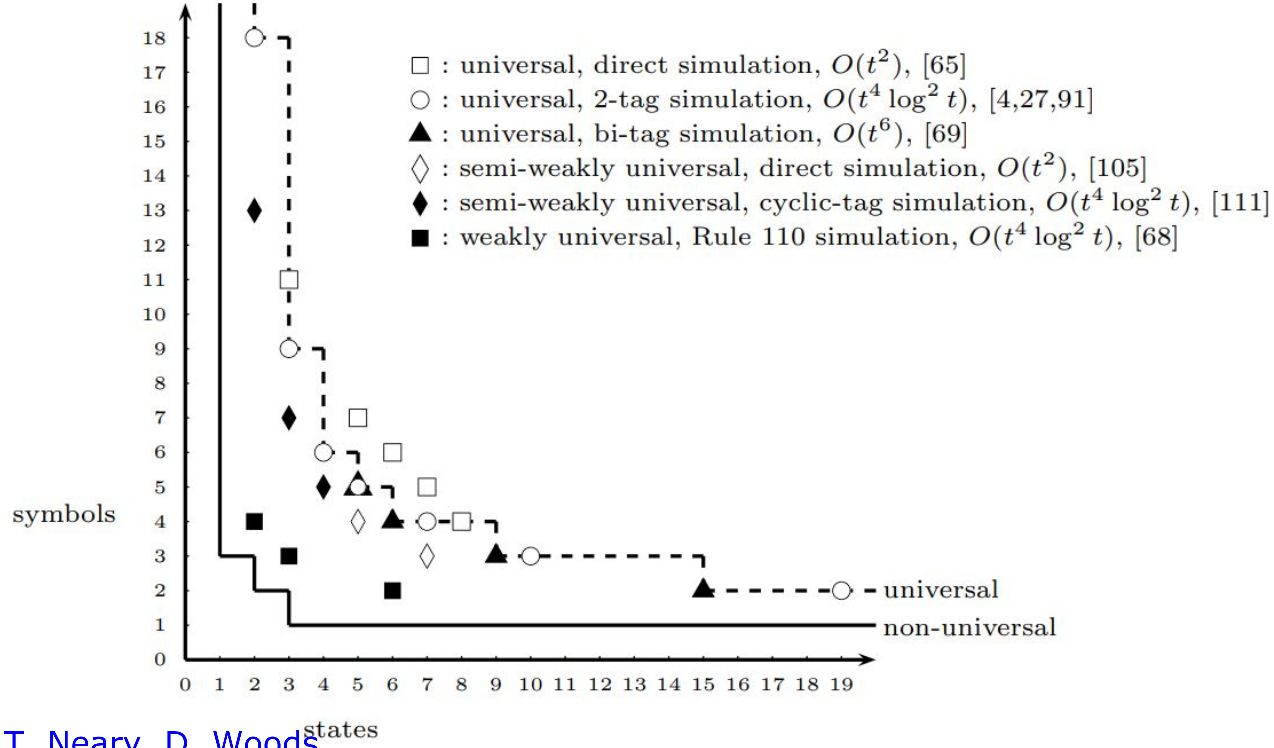
https://github.com/tcosmo/alang

Two major properties

1. There exists Turing machines that can compute **anything**: they are called Universal Turing machines.

2. There exists functions that no Turing machine can compute.

Universal Turing Machines



6 instructions and 6 symbols is all it takes!

T. Neary, D. Woods.
The complexity of small universal Turing machines: A survey. SOFSEM 2012.
https://arxiv.org/abs/1110.2230

Uncomputable Functions

- We say that a function f: N → N, is **computable** if there is a Turing machine such that, starting with ‘x’ on its tape will compute ‘f(x)’ and write it on its tape.

- The set of all Turing machines is **countable**.

- The set of all functions f: N → N is **not countable**.

- Therefore, there must exists functions that cannot be computed.

Uncomputable Functions

- We say that a function f: N → N, is **computable** if there is a Turing machine such that, starting with ‘x’ on its tape will compute ‘f(x)’ and write it on its tape.

- The set of all Turing machines is **countable**.

- The set of all functions f: N → N is **not countable**.

- Therefore, there must exists functions that cannot be computed.

Can we exhibit one?

The Halting Problem, Alan Turing, 1936

Is there a program `Halt` such that:

- `Halt(M,i)` = 1 iff program M halts on i
- `Halt(M,i)` = 0 otherwise

The Halting Problem, Alan Turing, 1936

Is there a program `Halt` such that:

- `Halt(M,i)` = 1 iff program M halts on i
- `Halt(M,i)` = 0 otherwise

For instance we have:

Halt(CopyMachine,'00101') = 1
Halt(WhileTrue,'0') = 0
...

The Halting Problem, Alan Turing, 1936

Let suppose that `Halt` exists.

Then let's build a new program Contradiction that takes as input a program M:

```
Contradiction(M):
        if  Halt(M,M):
            while true:
                continue
        else:
            return
```

The Halting Problem, Alan Turing, 1936

Let suppose that `Halt` exists.

Then let's build a new program Contradiction that takes as input a program M:

```
Contradiction(M):
        if  Halt(M,M):
            while true:
                continue
        else:
            return
```

Does `Contradiction(Contradiction)` halt?

- If it halts, it does not halt
- If it does not halt, it halts

The Halting Problem, Alan Turing, 1936

Let suppose that `Halt` exists.

Then let’s build a new program Contradiction that takes as input a program M:

```
Contradiction(M):
        if  Halt(M,M):
            while true:
                continue
        else:
            return
```

Does `Contradiction(Contradiction)` halt?

- If it halts, it does not halt
- If it does not halt, it halts

Contradiction!!

`Halt` does not exist

Making the link with mathematical knowledge

What is a proof?

- A finite object
- Which starts from **axioms** and applies rules of logic
- In order to reach a logically valid conclusion

Making the link with mathematical knowledge

What are axioms?

Robinson's axioms of arithmetic

1. $(\forall x)\neg Sx = 0.$
2. $(\forall x)(\forall y)\big[Sx = Sy \rightarrow x = y\big].$
3. $(\forall x)x + 0 = x.$
4. $(\forall x)(\forall y)x + Sy = S(x + y).$
5. $(\forall x)x \cdot 0 = 0.$
6. $(\forall x)(\forall y)x \cdot Sy = (x \cdot y) + x.$

The language of arithmetic is:

- The symbol 0
- The successor function S
- The addition function +
- The multiplication function ×
- The order relation <

Example: The number "1" is represented by S0, the number "2" is represented by SS0, etc..

Making the link with mathematical knowledge

What are axioms?

Robinson's axioms of arithmetic

1. $(\forall x)\neg Sx = 0.$
2. $(\forall x)(\forall y)\left[Sx = Sy \rightarrow x = y\right].$
3. $(\forall x)x + 0 = x.$
4. $(\forall x)(\forall y)x + Sy = S(x + y).$
5. $(\forall x)x \cdot 0 = 0.$
6. $(\forall x)(\forall y)x \cdot Sy = (x \cdot y) + x.$

The axioms say that:

1. $x + 1 = 0$ has no solution in N
2. $x + 1 = y + 1 \Rightarrow x = y$
3. $x + 0 = x$
4. $x + (y+1) = (x+y) + 1$
5. $x * 0 = 0$
6. $x * (y+1) = (x*y) + x$

Making the link with mathematical knowledge

A proof that 1+1 = 2

Robinson's axioms of arithmetic

1. $(\forall x)\neg Sx = 0.$
2. $(\forall x)(\forall y)\left[Sx = Sy \rightarrow x = y\right].$
3. $(\forall x)x + 0 = x.$
4. $(\forall x)(\forall y)x + Sy = S(x + y).$
5. $(\forall x)x \cdot 0 = 0.$
6. $(\forall x)(\forall y)x \cdot Sy = (x \cdot y) + x.$

$(\forall x)(\forall y)x + Sy = S(x + y)$	(axiom 4)
$S0 + S0 = S(S0 + 0)$	(instantiation)
$(\forall x)x + 0 = x$	(axiom 3)
$S0 + 0 = S0$	(instantiation)
$S0 + S0 = SS0$	(replacement)

Making the link with mathematical knowledge

A proof that 1+1 = 2

Robinson's axioms of arithmetic

1. $(\forall x)\neg Sx = 0.$
2. $(\forall x)(\forall y)\left[Sx = Sy \to x = y\right].$
3. $(\forall x)x + 0 = x.$
4. $(\forall x)(\forall y)x + Sy = S(x + y).$
5. $(\forall x)x \cdot 0 = 0.$
6. $(\forall x)(\forall y)x \cdot Sy = (x \cdot y) + x.$

$(\forall x)(\forall y)x + Sy = S(x + y)$	(axiom 4)
$S0 + S0 = S(S0 + 0)$	(instantiation)
$(\forall x)x + 0 = x$	(axiom 3)
$S0 + 0 = S0$	(instantiation)
$S0 + S0 = SS0$	(replacement)

1 + 1 = 2!!

Axioms *describe*

1. $(\forall x)\neg Sx = 0.$
2. $(\forall x)(\forall y)\left[Sx = Sy \rightarrow x = y\right].$
3. $(\forall x)x + 0 = x.$
4. $(\forall x)(\forall y)x + Sy = S(x + y).$
5. $(\forall x)x \cdot 0 = 0.$
6. $(\forall x)(\forall y)x \cdot Sy = (x \cdot y) + x.$

$$\mathbb{N}$$

Axioms and proofs are part of **Knowledge**

Mathematical objects are part of **Reality**

Axioms *describe*

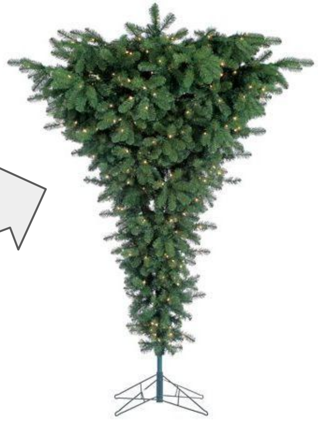
- A tree is tall
- A tree's foliage is green
- A tree's trunk is brown

Axioms and proofs are part of **Knowledge**

Objects are part of **Reality**

Axioms *describe*

1) We can end up describing things which are not what we mean by “tree”.



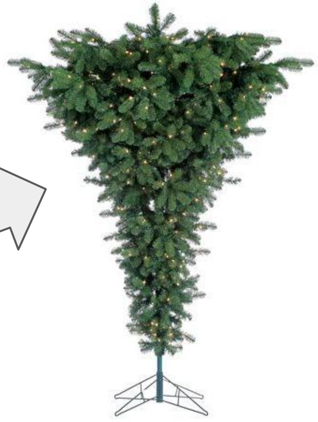
- A tree is tall
- A tree’s foliage is green
- A tree’s trunk is brown

Axioms and proofs are part of **Knowledge**

Objects are part of **Reality**

Axioms *describe*

1) We can end up describing things which are not what we mean by “tree”.



- A tree is tall
- A tree’s foliage is green
- A tree’s trunk is brown

2) There are some properties about trees that we won’t be able to deduce from our primitive description.

Axioms and proofs are part of **Knowledge**

Objects are part of **Reality**

Axioms *describe*

1. $(\forall x)\neg Sx = 0.$
2. $(\forall x)(\forall y)\left[Sx = Sy \rightarrow x = y\right].$
3. $(\forall x)x + 0 = x.$
4. $(\forall x)(\forall y)x + Sy = S(x + y).$
5. $(\forall x)x \cdot 0 = 0.$
6. $(\forall x)(\forall y)x \cdot Sy = (x \cdot y) + x.$

Non standard models of arithmetic

$$\mathbb{N}$$

Undecidable statements, here the commutativity of addition for instance.

Axioms and proofs are part of **Knowledge**

Mathematical objects are part of **Reality**

Simulating Turing machines with numbers

The parity machine:

- Takes a binary input
- Has three states {even, odd, halt}
- Decides if the number of 1s in the input is odd or even

```
Parity:
    even:
        if  read(0):
            goto even
        if read(1):
            goto odd
        if  read(#):
            goto halt
    odd:
        if read(0):
            goto odd
        if read(1):
            goto even
        if read(#):
            goto halt
    halt:
        Halt
```

Simulating Turing machines with numbers

Primes Encoding: $2^{\text{instruction number}}\ 3^{\text{head position}}\ 5^{\text{tape}_0}\ 7^{\text{tape}_1}\ 11^{\text{tape}_2}\ \ldots$

Example of a valid **trace** starting on input **`010`**:

$$x_0 = 2^0\ 3^0\ 5^0\ 7^1\ 11^0\ 13^2\ 17^0\ 19^0\ \ldots$$

Head

Simulating Turing machines with numbers

Primes Encoding: $2^{\text{instruction number}}\ 3^{\text{head position}}\ 5^{\text{tape}_0}\ 7^{\text{tape}_1}\ 11^{\text{tape}_2}\ \ldots$

Example of a valid **trace** starting on input **`010`**:

$$x_0 = 2^0\ 3^0\ 5^0\ 7^1\ 11^0\ 13^2\ 17^0\ 19^0\ \ldots$$

$$x_1 = 2^0\ 3^1\ 5^0\ 7^1\ 11^0\ 13^2\ 17^0\ 19^0\ \ldots$$

Head

Simulating Turing machines with numbers

Primes Encoding: $2^{\text{instruction number}}\ 3^{\text{head position}}\ 5^{\text{tape}_0}\ 7^{\text{tape}_1}\ 11^{\text{tape}_2}\ \ldots$

Example of a valid **trace** starting on input **`010`:**

$$x_0 = 2^0\ 3^0\ 5^0\ 7^1\ 11^0\ 13^2\ 17^0\ 19^0\ \ldots$$

$$x_1 = 2^0\ 3^1\ 5^0\ 7^1\ 11^0\ 13^2\ 17^0\ 19^0\ \ldots$$

$$x_2 = 2^1\ 3^2\ 5^0\ 7^1\ 11^0\ 13^2\ 17^0\ 19^0\ \ldots$$

Head

Simulating Turing machines with numbers

Primes Encoding: $2^{\text{instruction number}}\ 3^{\text{head position}}\ 5^{\text{tape}_0}\ 7^{\text{tape}_1}\ 11^{\text{tape}_2}\ \ldots$

Example of a valid **trace** starting on input **\`010\`:**

$$x_0 = 2^0\ 3^0\ 5^0\ 7^1\ 11^0\ 13^2\ 17^0\ 19^0\ \ldots$$
$$x_1 = 2^0\ 3^1\ 5^0\ 7^1\ 11^0\ 13^2\ 17^0\ 19^0\ \ldots$$
$$x_2 = 2^1\ 3^2\ 5^0\ 7^1\ 11^0\ 13^2\ 17^0\ 19^0\ \ldots$$
$$x_3 = 2^1\ 3^3\ 5^0\ 7^1\ 11^0\ 13^2\ 17^0\ 19^0\ \ldots$$

Head

Simulating Turing machines with numbers

Primes Encoding: $2^{\text{instruction number}}\ 3^{\text{head position}}\ 5^{\text{tape}_0}\ 7^{\text{tape}_1}\ 11^{\text{tape}_2}\ \ldots$

Example of a valid **trace** starting on input **`010`:**

$$x_0 = 2^0\ 3^0\ 5^0\ 7^1\ 11^0\ 13^2\ 17^0\ 19^0\ \ldots$$

$$x_1 = 2^0\ 3^1\ 5^0\ 7^1\ 11^0\ 13^2\ 17^0\ 19^0\ \ldots$$

$$x_2 = 2^1\ 3^2\ 5^0\ 7^1\ 11^0\ 13^2\ 17^0\ 19^0\ \ldots$$

$$x_3 = 2^1\ 3^3\ 5^0\ 7^1\ 11^0\ 13^2\ 17^0\ 19^0\ \ldots$$

$$x_4 = 2^2\ 3^3\ 5^0\ 7^1\ 11^0\ 13^2\ 17^0\ 19^0\ \ldots$$

Halt!

Simulating Turing machines with numbers

Primes Encoding: $2^{\text{instruction number}}\ 3^{\text{head position}}\ 5^{\text{tape}_0}\ 7^{\text{tape}_1}\ 11^{\text{tape}_2}\ \ldots$

Example of a valid **trace** starting on input **`010`**:

$$
\begin{aligned}
x_0 &= 2^0\ 3^0\ 5^0\ 7^1\ 11^0\ 13^2\ 17^0\ 19^0\ \ldots \\
x_1 &= 2^0\ 3^1\ 5^0\ 7^1\ 11^0\ 13^2\ 17^0\ 19^0\ \ldots \\
x_2 &= 2^1\ 3^2\ 5^0\ 7^1\ 11^0\ 13^2\ 17^0\ 19^0\ \ldots \\
x_3 &= 2^1\ 3^3\ 5^0\ 7^1\ 11^0\ 13^2\ 17^0\ 19^0\ \ldots \\
x_4 &= 2^2\ 3^3\ 5^0\ 7^1\ 11^0\ 13^2\ 17^0\ 19^0\ \ldots
\end{aligned}
$$

Halt!

Robinson’s arithmetic is powerful enough to “check” all these steps!

Simulating Turing machines with numbers

Example of a valid **trace** for the parity machine starting on input **`010`**:

$$x_0 = 2^0 \; 3^0 \, 5^0 \, 7^1 \, 11^0 \, 13^2 \, 17^0 \, 19^0 \; \ldots$$
$$x_1 = 2^0 \; 3^1 \, 5^0 \, 7^1 \, 11^0 \, 13^2 \, 17^0 \, 19^0 \; \ldots$$
$$x_2 = 2^1 \; 3^2 \, 5^0 \, 7^1 \, 11^0 \, 13^2 \, 17^0 \, 19^0 \; \ldots$$
$$x_3 = 2^1 \; 3^3 \, 5^0 \, 7^1 \, 11^0 \, 13^2 \, 17^0 \, 19^0 \; \ldots$$
$$x_4 = 2^2 \; 3^3 \, 5^0 \, 7^1 \, 11^0 \, 13^2 \, 17^0 \, 19^0 \; \ldots$$

We can formulate the Halting problem of our parity solving machine in the language of arithmetic:

$$\begin{aligned} \mathrm{doesParityMachineHalt}(i) \equiv \quad & \exists k \, \exists x_0, \ldots, x_k \\ & \mathrm{isTapeContentMatching}(x_0, i) \\ & \wedge (\forall i < k) \; \mathrm{isValidParityMachineTransition}(x_i, x_{i+1}) \\ & \wedge \mathrm{hasParityMachineHalted}(x_k) \end{aligned}$$

Simulating Turing machines with numbers

We can formulate the Halting problem of **any machine** in the language of arithmetic:

$$\begin{aligned}\text{doesMachineHalt}(M, i) \equiv \quad & \exists k\ \exists x_0, \ldots, x_k \\ & \text{isTapeContentMatching}(x_0, i) \\ & \wedge (\forall i < k)\ \text{isValidMachineTransition}(M, x_i, x_{i+1}) \\ & \wedge\ \text{hasMachineHalted}(M, x_k)\end{aligned}$$

Theorem
The machine \`M\` halts on input \`i\` if and only if there is a proof of the statement ‘doesMachineHalt(M,i)’ from Robison’s axioms.

Back to the Halting Problem

Theorem
The machine `M` halts on input `i` if and only if there is a proof of the statement 'doesMachineHalt(M,i)' from Robison's axioms.

If any true statement was provable using Robison's axioms, we could solve the Halting problem:

- Enumerate all proofs that use Robison's axioms until either:
  - You find a proof that concludes `doesMachineHalt(M,i)` return true
  - You find a proof that concludes `not doesMachineHalt(M,i)` return false

That **contradicts** the uncomputability of the Halting Problem!

Hence, there must exists a statement about the natural numbers that is **true** but that **we cannot prove** using Robinson's axioms!

First Incompleteness Theorem

First Incompleteness Theorem (Kurt Gödel, 1931)

*For any **consistent** and **computable** set of axioms A expressed in the language of arithmetic, there exists a statement that is true in the natural numbers but that cannot be proved from this set of axioms.*

First Incompleteness Theorem

First Incompleteness Theorem (Kurt Gödel, 1931)

*For any **consistent** and **computable** set of axioms A expressed in the language of arithmetic, there exists a statement that is true in the natural numbers but that cannot be proved from this set of axioms.*

- If A is weaker than Robinson's axioms, i.e. there is a Robinson axiom it cannot prove, just take that statement as the unprovable true statement.

- If A is stronger than Robinson's axioms:

    - You have enough logic to run Turing machines
    - Because A is **computable** you still can enumerate all proofs from A

First Incompleteness Theorem

First Incompleteness Theorem (Kurt Gödel, 1931)

*For any **consistent** and **computable** set of axioms A expressed in the language of arithmetic, there exists a statement that is true in the natural numbers but that cannot be proved from this set of axioms.*

- If A is weaker than Robinson's axioms, i.e. there is a Robinson axiom it cannot prove, just take that statement as the unprovable true statement.
- If A is stronger than Robinson's axioms:
  - You have enough logic to run Turing machines
  - Because A is **computable** you still can enumerate all proofs from A

Christopher C. Leary. Lars Kristiansen. A Friendly Introduction to Mathematical Logic (2nd. ed.). 2015. Geneso, NY.

First Incompleteness Theorem

Non constructively we have reached the conclusion that, for any computable set of axioms A stronger than Robinson:

If A is consistent then, there exists a machine `M` and input `i` such that neither statements can be proven from A:

- `doesMachineHalt(M,i)`, meaning “The machine M halts on input i”
- `not doesMachineHalt(M,i)` meaning “The machine M does not halt on input i”

First Incompleteness Theorem

Non constructively we have reached the conclusion that, for any computable set of axioms A stronger than Robinson:

If A is consistent then, there exists a machine `M` and input `i` such that neither statements can be proven from A:

- `doesMachineHalt(M,i)`
- `not doesMachineHalt(M,i)`

BUT

First Incompleteness Theorem

Non constructively we have reached the conclusion that, for any computable set of axioms A stronger than Robinson:

If A is consistent then, there exists a machine `M` and input `i` such that neither statements can be proven from A:

- `doesMachineHalt(M,i)`
- `not doesMachineHalt(M,i)`

BUT, `doesMachineHalt(M,i)` is an existential-only statement !

$$\begin{aligned}\text{doesMachineHalt}(M,i) \equiv \quad & \exists k\ \exists x_0,\ldots,x_k \\ & \text{isTapeContentMatching}(x_0,i) \\ & \wedge (\forall i < k)\ \text{isValidMachineTransition}(M,x_i,x_{i+1}) \\ & \wedge \text{hasMachineHalted}(M,x_k)\end{aligned}$$

First Incompleteness Theorem

Non constructively we have reached the conclusion that, for any computable set of axioms A stronger than Robinson:

If A is consistent then, there exists a machine \`M\` and input \`i\` such that neither statements can be proven from A:

- ~~\`doesMachineHalt(M,i)\`~~
- \`not doesMachineHalt(M,i)\`

BUT, \`doesMachineHalt(M,i)\` is an existential-only statement !
SO, if you cannot prove it…. Then it is **false**. Hence, \`not doesMachineHalt(M,i)\` **holds**.

$$\begin{aligned}\mathrm{doesMachineHalt}(M,i) \equiv \quad & \exists k\, \exists x_0, \ldots, x_k \\ & \mathrm{isTapeContentMatching}(x_0, i) \\ & \wedge (\forall i < k)\ \mathrm{isValidMachineTransition}(M, x_i, x_{i+1}) \\ & \wedge \mathrm{hasMachineHalted}(M, x_k)\end{aligned}$$

First Incompleteness Theorem

Non constructively we have reached the conclusion that, for any computable set of axioms A stronger than Robinson:

If A is consistent then, there exists a machine `M` and input `i` such that neither statements can be proven from A:

- ~~`doesMachineHalt(M,i)`~~
- `not doesMachineHalt(M,i)`

BUT, `doesMachineHalt(M,i)` is an existential-only statement !
SO, if you cannot prove it…. Then it is **false**. Hence, `not doesMachineHalt(M,i)` **holds**.

BUT WAIT, I just gave a proof that the machine does not halt!!

First Incompleteness Theorem

Non constructively we have reached the conclusion that, for any computable set of axioms A stronger than Robinson:

If A is consistent then, there exists a machine `M` and input `i` such that neither statements can be proven from A:

- ~~`doesMachineHalt(M,i)`~~
- `not doesMachineHalt(M,i)`

BUT, `doesMachineHalt(M,i)` is an existential-only statement !
SO, if you cannot prove it…. Then it is **false**. Hence, `not doesMachineHalt(M,i)` **holds**.

BUT WAIT, I just gave a proof that the machine does not halt!!

First Incompleteness Theorem

Non constructively we have reached the conclusion that, for any computable set of axioms A stronger than Robinson:

If A is consistent then, there exists a machine `M` and input `i` such that neither statements can be proven from A:

- ~~`doesMachineHalt(M,i)`~~
- `not doesMachineHalt(M,i)`

BUT, `doesMachineHalt(M,i)` is an existential-only statement !
SO, if you cannot prove it…. Then it is **false**. Hence, `not doesMachineHalt(M,i)` **holds**.

BUT WAIT, I just gave a proof that the machine does not halt!!

Conclusion: the set of axioms A cannot prove its own consistency!

Second Incompleteness Theorem

Second Incompleteness Theorem (Kurt Gödel, 1931)

*For any **consistent** and **computable** set of axioms A expressed in the language of arithmetic, which is at least as strong as **Peano's axioms** then the following statement is not provable in A:*

$$\neg \mathrm{isTheoremUsingAxiomsA}(0 = S0)$$

Peano's axioms = Robinson's axioms + induction

Conclusion: you cannot prove that it is not possible to prove 0 = 1 from Peano's axiom if you limit yourself to using Peano's axioms.

Second Incompleteness Theorem

Second Incompleteness Theorem (Kurt Gödel, 1931)

*For any **consistent** and **computable** set of axioms A expressed in the language of arithmetic, which is at least as strong as **Peano's axioms** then the following statement is not provable in A:*

$$\neg \mathrm{isTheoremUsingAxiomsA}(0 = S0)$$

Peano's axioms = Robinson's axioms + induction

Conclusion: you cannot prove that it is not possible to prove 0 = 1 from Peano's axiom if you limit yourself to using Peano's axioms.

You can prove Peano's consistency using ZFC axioms. But you wont prove ZFC's. Etc...

Incomprehensible Machines

- A machine that iterates all proofs in Peano/ZFC and halts if and only if it finds a proof of 0 = 1
  - People have actually built such a machine: 7,910 instructions
- There is a 27-instruction Turing machine that halts iff Goldbach Conjecture is true
- There is a 744-instruction Turing machine that halts iff Riemann Hypothesis is true

A. Yedidia and S. Aaronson
A Relatively Small Turing Machine Whose Behavior Is Independent of Set Theory.
https://arxiv.org/abs/1605.04343
S. Aaronson
The Busy Beaver frontier. https://www.scottaaronson.com/papers/bb.pdf

Incomprehensible Machines

- Scott Aaronson conjectures:
  - There is a 10-instruction Turing machine whose halting problem is independent of Peano's axioms
  - There is a 20-instruction Turing machine whose halting problem is independent of ZFC's axioms

Good contenders are Collatz-like:

$$g(x) := \begin{cases} \frac{5x+18}{3} & \text{if } x \equiv 0 \pmod{3} \\ \frac{5x+22}{3} & \text{if } x \equiv 1 \pmod{3} \\ \perp & \text{if } x \equiv 2 \pmod{3} \end{cases}$$

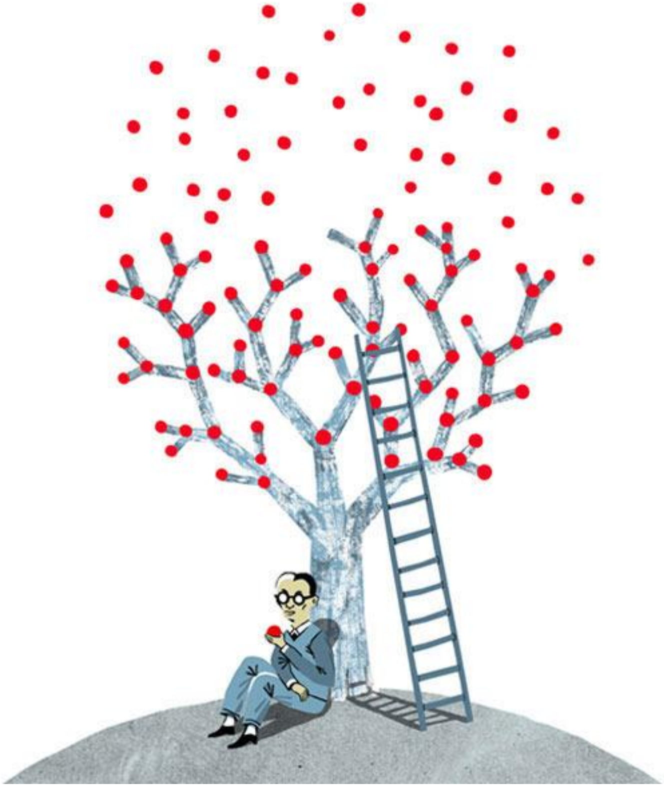
H. Marxen and J. Buntrock. Attacking the Busy Beaver 5. 1990. EATCS.

Incomprehensible Machines

```python
import itertools

def Collatz(x):
    if x%2 == 0:
        return x//2
    return 3*x + 1

def apply_Collatz_function(x,n_times):
    for _ in range(n_times):
        x = Collatz(x)
    return x

n = 1
isRunning = True
while isRunning:
    for binary_string in itertools.product(["0","1"],repeat=n):
        starting_point = int("".join(binary_string),2)
        ending_point = apply_Collatz_function(starting_point,len(binary_string))

        if starting_point == ending_point and starting_point not in [0,1,2,4]:
            isRunning = False

    n += 1
```

Questions :)?

Alan Turing



Kurt Gödel