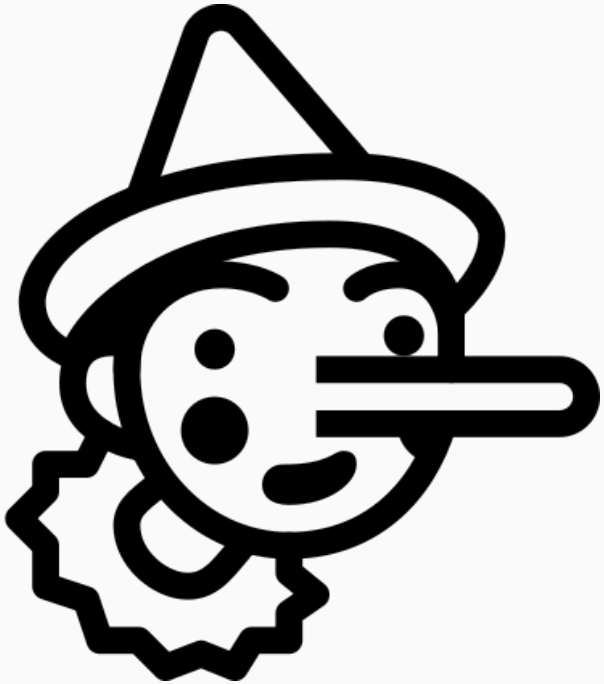
Introduction to computability

6CCS3COM Computational Models

Josh Murphy

Warm up

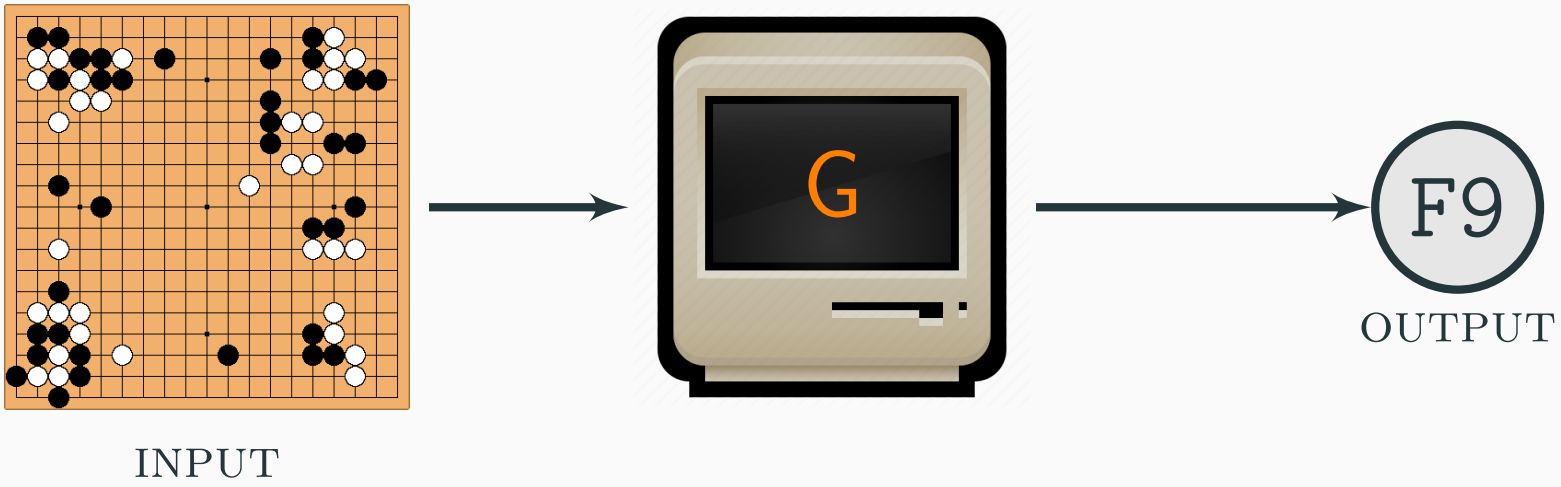


- Pinocchio's nose grows whenever he says something that is not true.
- Will Pinocchio's nose grow if he says "My nose will grow now"?

Contents

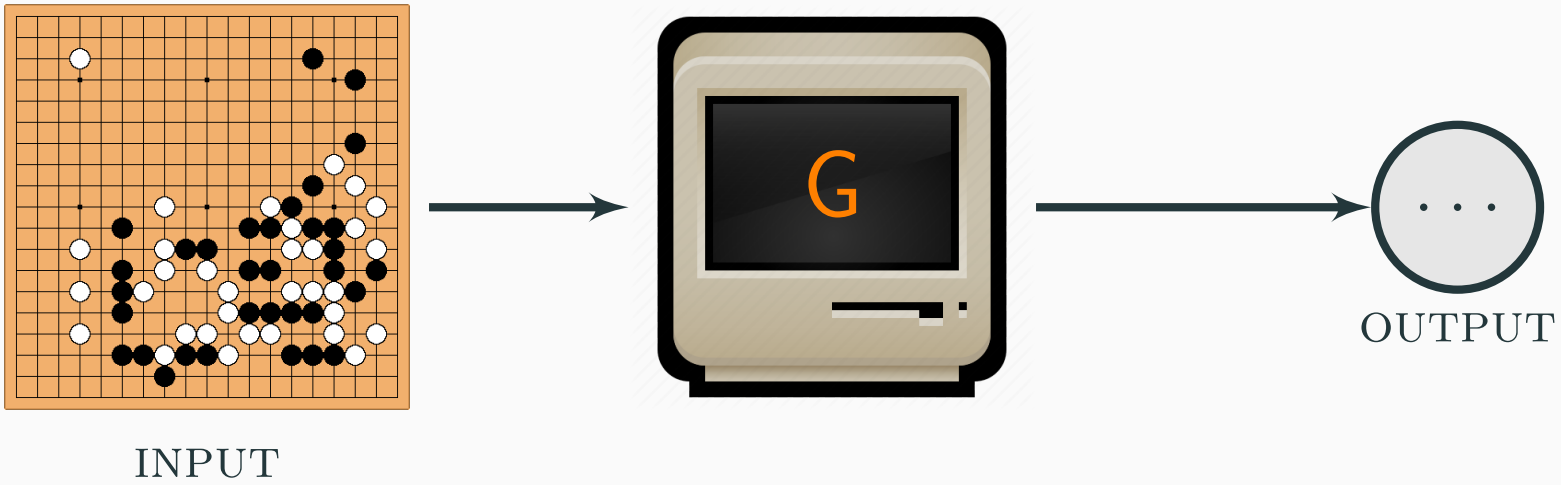
- What is the **Halting Problem**?
- Representing programs as **partial functions**.

A program for playing go



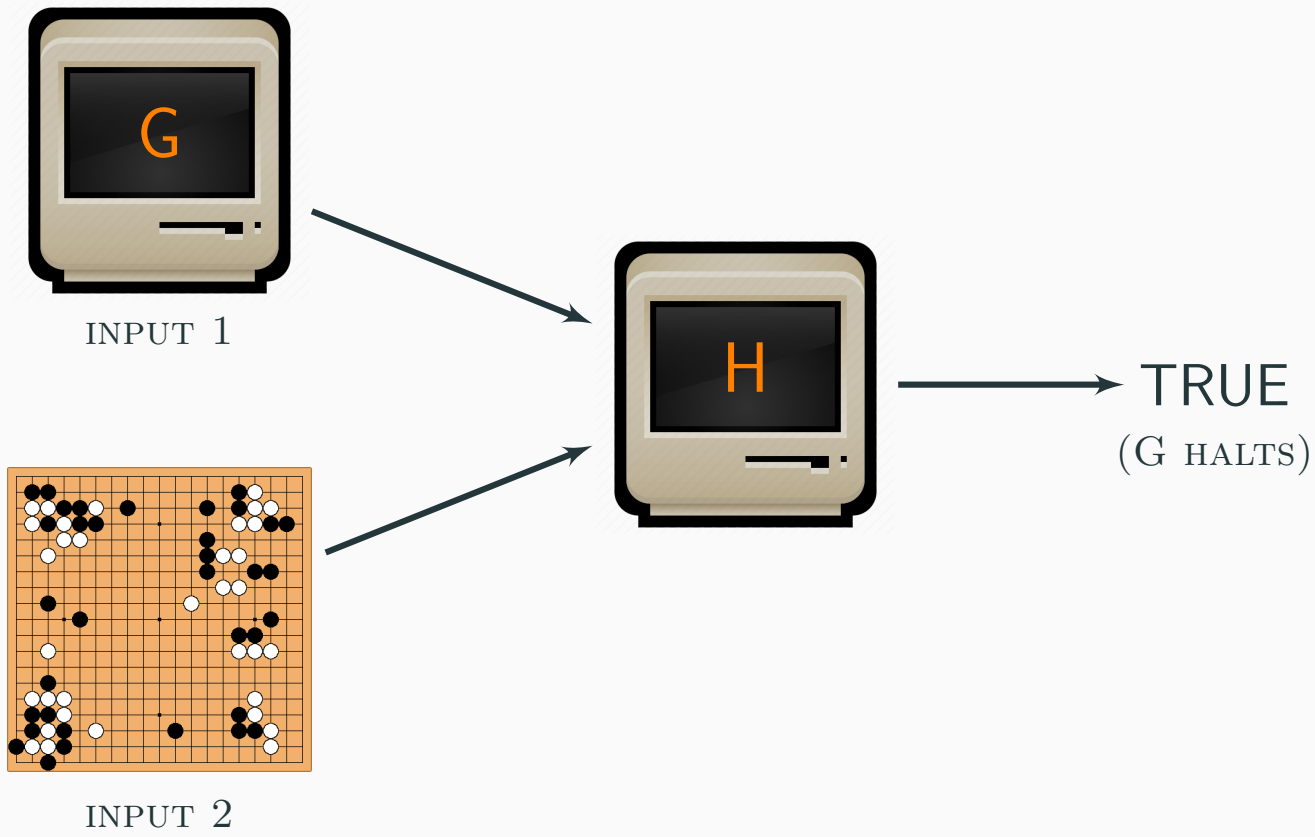
- **G** is a program that takes a Go state and outputs the optimal move.

Programs do not always halt (finish in finite time)



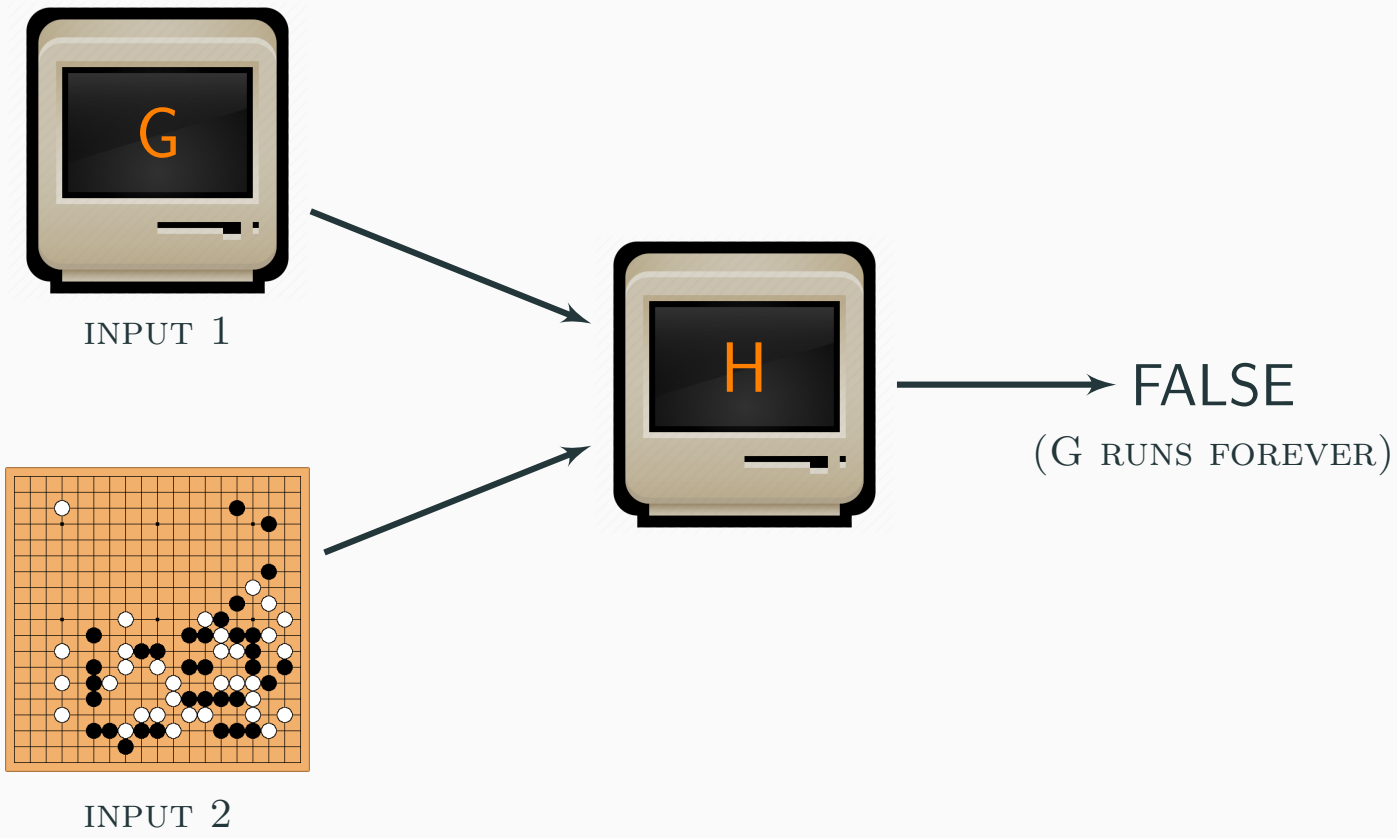
- The program sometimes gets stuck and will never return an output.

Will G eventually give an output?



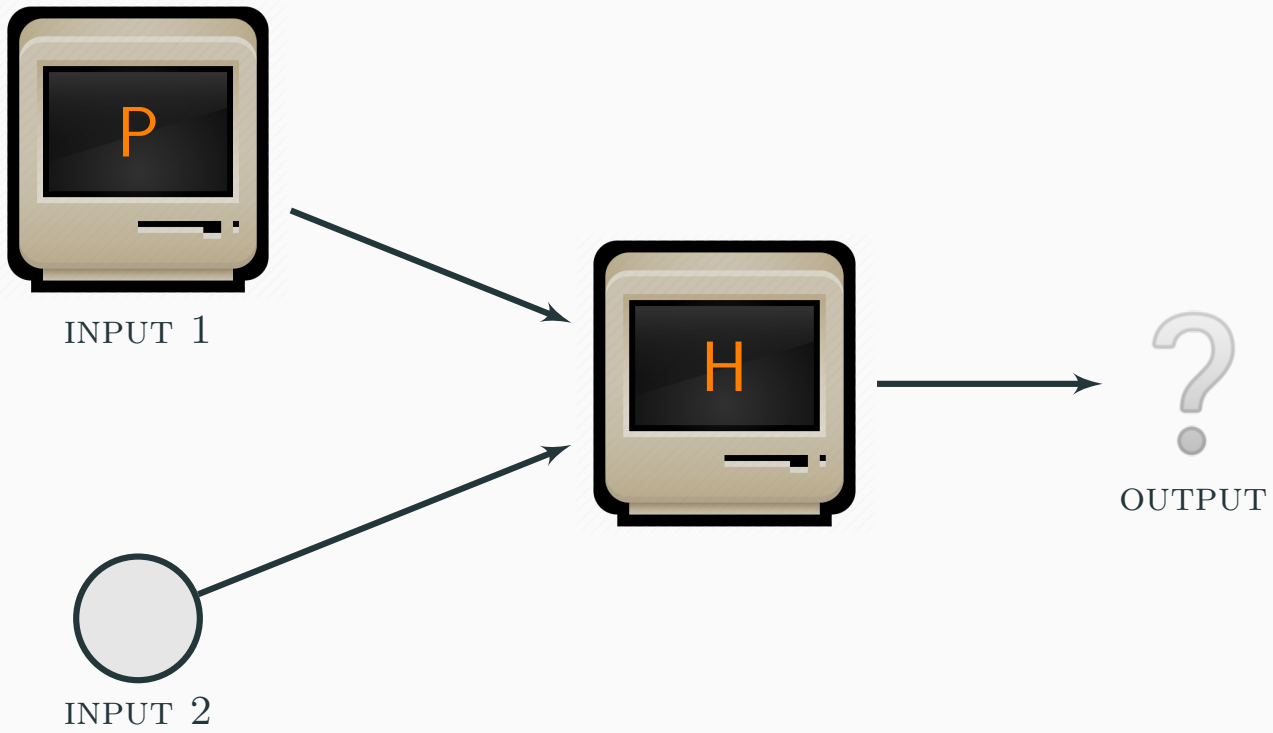
- **H** is a program that takes **G** and a Go state as inputs and outputs true iff **G** will halt on that Go state (false otherwise).

Will G eventually give an output?



- **H** is a program that takes **G** and a Go state as inputs and outputs true iff **G** will halt on that Go state (false otherwise).

The Halting Problem



Is it possible for a program (**H**) to establish, from a description of an arbitrary program (**P**) and its input (**I**), whether the program will halt?

The Halting Problem

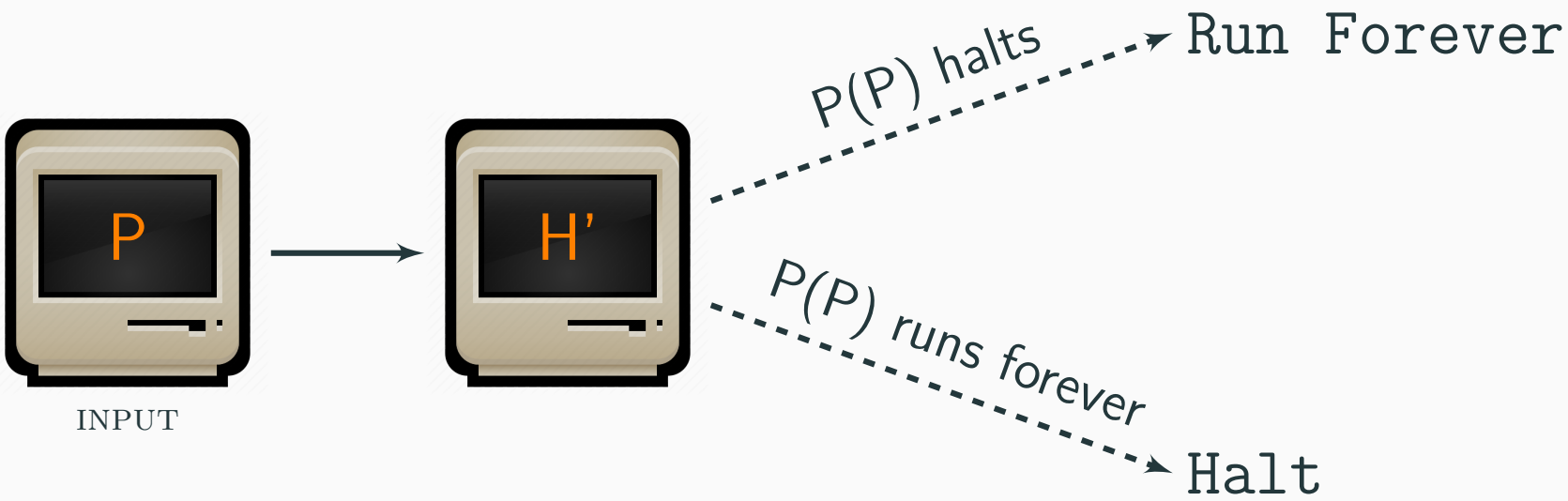
Algorithm 1

```
1: procedure H(P, I)
2:     if P(I) halts then
3:         return True;
4:     else
5:         return False;
6:     end if
7: end procedure
```

Is it possible for a program (**H**) to establish, from a description of an arbitrary program (**P**) and its input (**I**), whether the program will halt?

Let's assume H exists...

Let's make some small modifications to **H** to produce a new program **H'**.



- **H'** will take a program (**P**) as input.
- If **P**(**P**) halts then **H'** will run forever.
- If **P**(**P**) runs forever then **H'** will halt.

Modifications to H'

Let's make some small modifications to **H** to produce a new program **H'**.

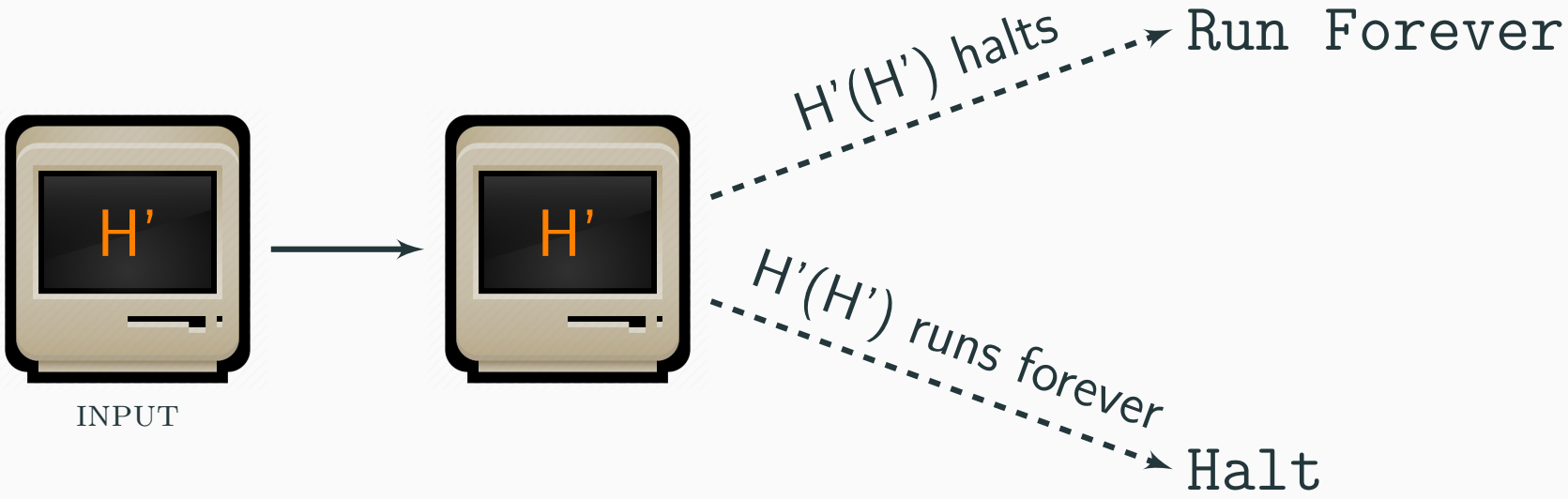
Algorithm 1

```
1: procedure H(P, I)
2:     if P(I) halts then
3:         return True;
4:     else
5:         return False;
6:     end if
7: end procedure
```

Algorithm 2

```
1: procedure H'(P)
2:     if P(P) halts then
3:         loop forever;
4:     else
5:         halt;
6:     end if
7: end procedure
```

Now feed H' into itself...



- If H'(H') halts then it runs forever...
- If H'(H') runs forever then it halts...
- **Contradictions**!

Paradox!

- By assuming that a program that solves the halting problem exists we have found a **contradiction**. Therefore, it's not possible for such a program to exists.
- If it is possible to design a program to answer a question within a finite time the problem is said to be **decidable**.
- We say that the halting problem is **undecidable**.
- The halting problem was one of the first problems to be shown to be undecidable. It demonstrated that there exists a **theoretical limit** to computability.
    - This proof was formalised by Turing, which required a formal model of a computer (**Turing machine**).

Other undecidable problems

- Solving Diophantine equations.
  - Hilbert's 10th Problem, proposed in 1900, shown to be undecidable in 1970.
- Magic: The Gathering strategies are undecidable.
  - https://arxiv.org/abs/1904.09828v2

Want to know more?

- **Logicomix:** An epic search for truth!

‘

Algorithms as partial functions

Algorithms

- Initially, algorithms were restrictively considered to be:
  - A **finite** description of a computation in terms of elementary instructions.
  - A **deterministic** procedure: the next step is uniquely defined.
  - Always **produces a result**, no matter that the input is (e.g. it halts).
- In this course we will focus on how computation mechanisms are expressed, and whether a problem has a computable solution or not, without caring about the efficiency of the algorithms.
- To allow us to represent unending computations (that do not always return a result), and to abstract away from the unnecessary efficiency details, we will consider programs to be **partial functions**.

Partial vs total functions

- Some expressions do not have a value:
  1. True + 3 is not well-defined (most type systems do not allow a boolean to be added to an integer).
  2. 10/0 does not have an arithmetic results.
  3. `fact(-1)` does not have a value where:
     `fact(n) = { if x=0 then 1, else x * fact(n-1) }`
- Expression 1 is a type error. Addition is a **total function** on the natural numbers because for any pair of natural numbers the result of addition is defined. But addititon is not defined for booleans.
- Expression 2 is a different type of problem. 10 and 0 are both natural numbers, but division by 0 is not defined. Division is a **partial function**.
- Expression 3 the computation does not produce an output, instead it loops forever. So the `fact` program is also a **partial function**.

First, some notation

- Let A and B be sets. Their **Cartseian product** is denoted by $A \times B$.
  - e.g. $\{a, b, c\} \times \{1, 2\} = \{(a, 1), (b, 1), (c, 1), (a, 2), (b, 2), (c, 2)\}$
- **Membership** of a set is denoted with $\in$. Examples:
  - $b \in \{a, b, c\}$
  - $d \notin \{a, b, c\}$
  - $\{c, 1\} \in A \times B$
  - $\{d, 3\} \notin A \times B$

Partial vs total functions (formally)

- A **partial function** f from A to B (abbreviated $f : A \to B$) is a subset of $A \times B$ such that if $(x, y) \in f$ and $(x, z) \in f$, then $y = z$.
  - In other words, a partial function from A to B associates to each element of A zero or one elements of B.
- A **total function** from A to B associates to each element of A exactly one element of B.
  - A is the **domain**
  - B is the **co-domain**
- Computer programs may not return an output (in the case where they loop forever or get stuck), so for some inputs there may be zero outputs, therefore computer programs are commonly represented as partial functions.

Summary

- The **Halting Problem** asks whether a program can exist that can detect whether a program halts or runs forever.
- We went through the intuition of a proof that shows such a program **cannot exist**, and thus demonstrates a theoretical limits to computability.
- We consider computer programs as **partial functions**, independant of any particular programming language or computer hardware.
- The next step is to investigate **Turing Machines**, which were used to construct the formal proof of the undecidability of the Halting Problem.