PHIL 222
Philosophical Foundations of Computer Science
Week 2, Thursday

Sept. 5, 2024

From Philosophy to Computer Science:
Entscheidungsproblem

So we have a system of proofs — e.g. the one by Frege (1879) — which consists of rules and (maybe) axioms.

So we have a system of proofs — e.g. the one by Frege (1879) — which consists of rules and (maybe) axioms.

We would want such a system to satisfy properties like:

- "Soundness": If there is a proof for an inference, the inference is "valid", meaning that there is no counterexample for it.

So we have a system of proofs — e.g. the one by Frege (1879) — which consists of rules and (maybe) axioms.

We would want such a system to satisfy properties like:

- "Soundness": If there is a proof for an inference, the inference is "valid", meaning that there is no counterexample for it.
- "Completeness": If an inference is valid, then there is a proof for it.

So we have a system of proofs — e.g. the one by Frege (1879) — which consists of rules and (maybe) axioms.

We would want such a system to satisfy properties like:

- "Soundness": If there is a proof for an inference, the inference is "valid", meaning that there is no counterexample for it.
- "Completeness": If an inference is valid, then there is a proof for it.

Many facts in mathematics can be proven from a small number of axioms.

So we have a system of proofs — e.g. the one by Frege (1879) — which consists of rules and (maybe) axioms.

We would want such a system to satisfy properties like:

- "Soundness": If there is a proof for an inference, the inference is "valid", meaning that there is no counterexample for it.
- "Completeness": If an inference is valid, then there is a proof for it.

Many facts in mathematics can be proven from a small number of axioms.

Hilbert and Ackermann (1928):

> The *Entscheidungsproblem* [decision problem] is solved when we know a procedure that allows for any given logical expression to decide by finitely many operations its validity or satisfiability [. . .]. The *Entscheidungsproblem* must be considered the main problem of mathematical logic.

- “Completeness”: If an inference is valid, then there is a proof for it.

- “Completeness”: If an inference is valid, then there is a proof for it.
  — Gödel (1930): Yes.

- “Completeness”: If an inference is valid, then there is a proof for it.
  — Gödel (1930): Yes.
- A proof can be verified mechanically.
  — What does “mechanically” really mean?

- “Completeness”: If an inference is valid, then there is a proof for it.
  — Gödel (1930): Yes.
- A proof can be verified mechanically.
  — What does “mechanically” really mean?
- Is there a set of axioms that achieves the following two goals?
  - All facts of arithmetic can be proven from it.
  - It can be mechanically checked whether a sentence is an axiom or not.

- “Completeness”: If an inference is valid, then there is a proof for it.
  — Gödel (1930): Yes.
- A proof can be verified mechanically.
  — What does “mechanically” really mean?
- Is there a set of axioms that achieves the following two goals?
  - All facts of arithmetic can be proven from it.
  - It can be mechanically checked whether a sentence is an axiom or not.

  — Gödel (1931): No, with a partial theory of “mechanically”.

- "Completeness": If an inference is valid, then there is a proof for it.
  — Gödel (1930): Yes.
- A proof can be verified mechanically.
  — What does "mechanically" really mean?
- Is there a set of axioms that achieves the following two goals?
  - All facts of arithmetic can be proven from it.
  - It can be mechanically checked whether a sentence is an axiom or not.

  — Gödel (1931): No, with a partial theory of "mechanically".
- Even though there is no perfect set of axioms, at least can it be mechanically decided whether an inference is valid or not?

- "Completeness": If an inference is valid, then there is a proof for it.
  — Gödel (1930): Yes.
- A proof can be verified mechanically.
  — What does "mechanically" really mean?
- Is there a set of axioms that achieves the following two goals?
  - All facts of arithmetic can be proven from it.
  - It can be mechanically checked whether a sentence is an axiom or not.

  — Gödel (1931): No, with a partial theory of "mechanically".
- Even though there is no perfect set of axioms, at least can it be mechanically decided whether an inference is valid or not?
  — Church (1936) and Turing (1936): No, with full theories of "mechanically". The *Entscheidungsproblem* is unsolvable.

There are (at least) three approaches to the theory of what can be done “mechanically”, or what computers can do — “computability theory”:

1. Gödel: “recursive functions”.
2. Church: “λ-calculus”.
3. Turing: “Turing machines”.

Turing Machines

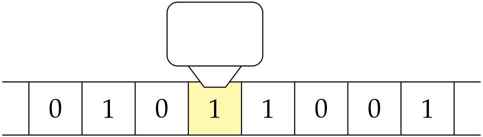
Advice

This is a technical part of the course that will be covered in the technical exercises and the midterm exam.

If anything here does not "click" in your mind,

- *Come to see me in office hours & appointments!*

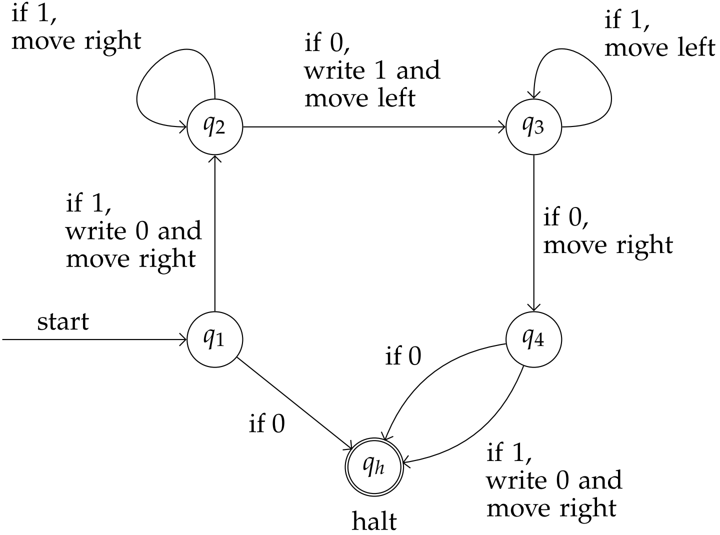
You are only expected to learn what the rule of the game is like. It is absolutely natural if it does not "click" in your mind for the first time you see it. But to resolve that situation, you need interactive help. Please, please help me help you.



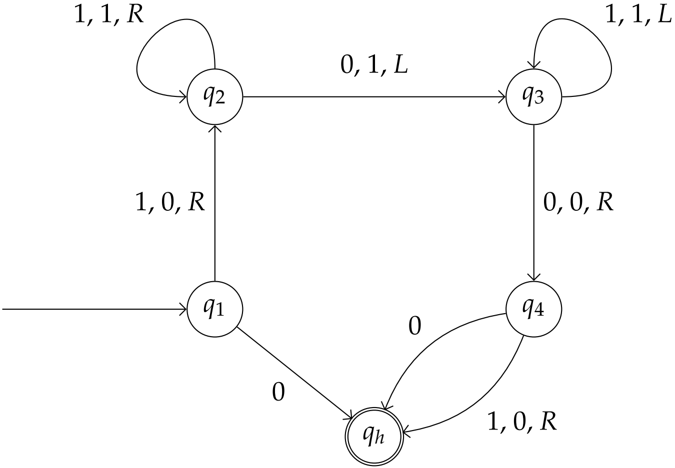
Two components:

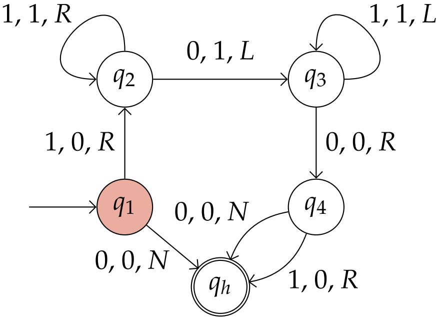
- A tape divided into squares. It may have infinitely many squares. Each square either is blank or contains a symbol from some finite alphabet.
- A machine called a **Turing machine**. At any time it is above one of the squares on the tape, and reads the symbol in it. It may then
  - (re)write a symbol or delete the symbol in the square; and
  - move one square to the left or right; or
  - halt.

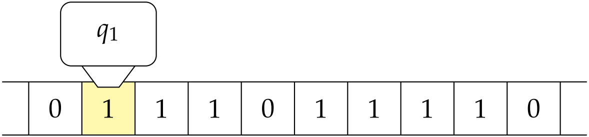
  The machine determines what to do at each step by its "(internal) state", its built-in instruction, and the symbol it reads.



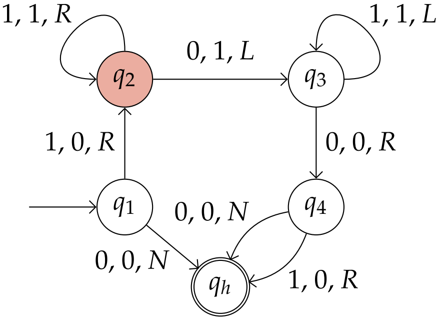
if 1,
move right
if 0,
write 1 and
move left
if 1,
move left
q_2
q_3
if 1,
write 0 and
move right
if 0,
move right
start
q_1
q_4
if 0
if 0
q_h
halt
if 1,
write 0 and
move right

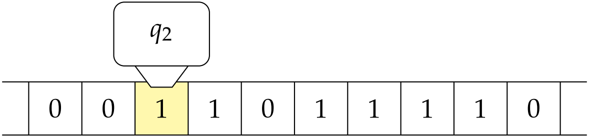

1, 1, R
0, 1, L
1, 1, L
q_2
q_3
1, 0, R
0, 0, R
q_1
q_4
0
0
q_h
1, 0, R



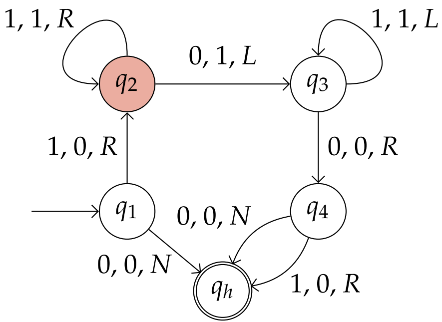


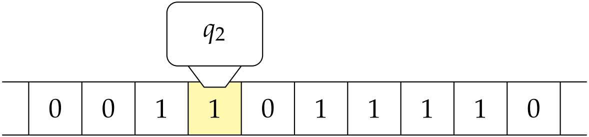
$t = 0$



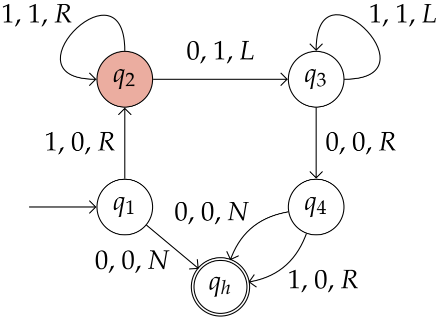


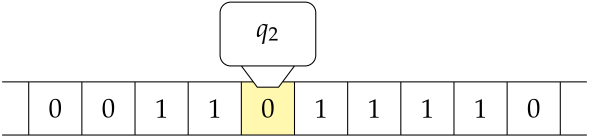
$t = 1$



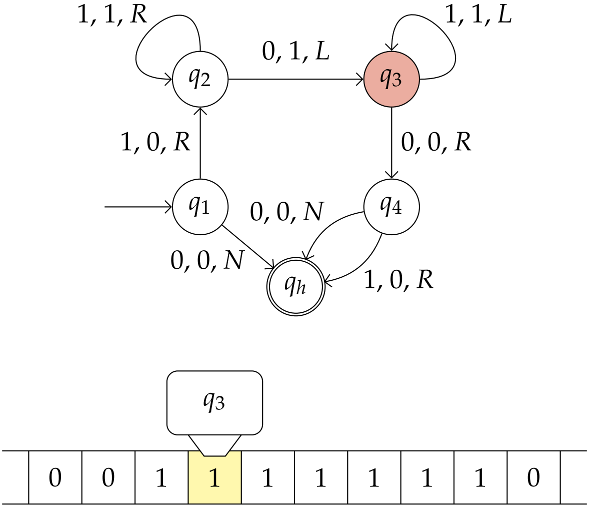


$t = 2$

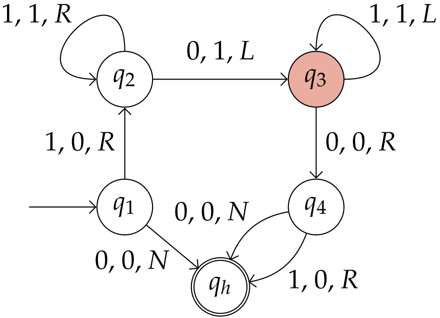


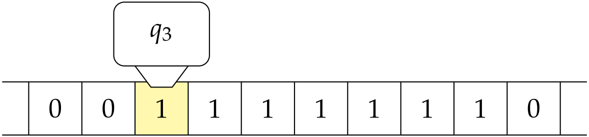


$t = 3$

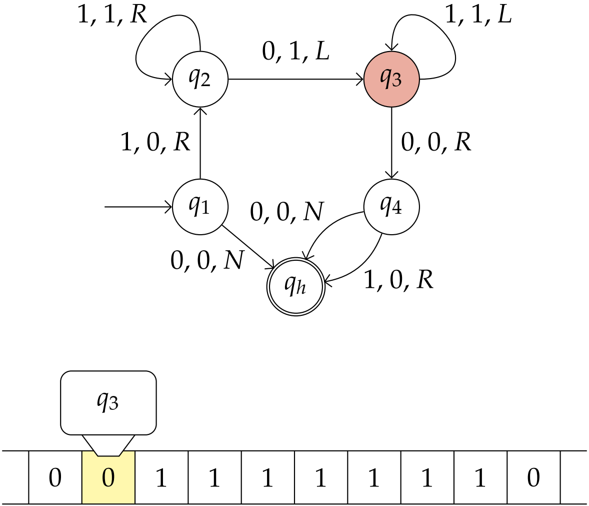


$t = 4$

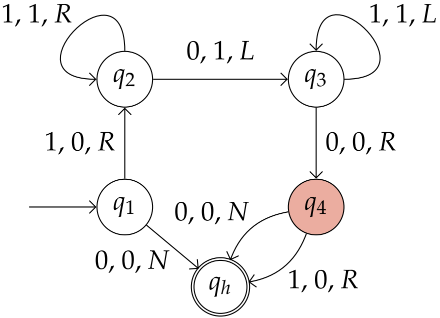


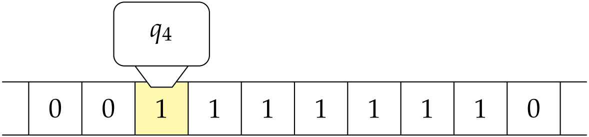


$t = 5$

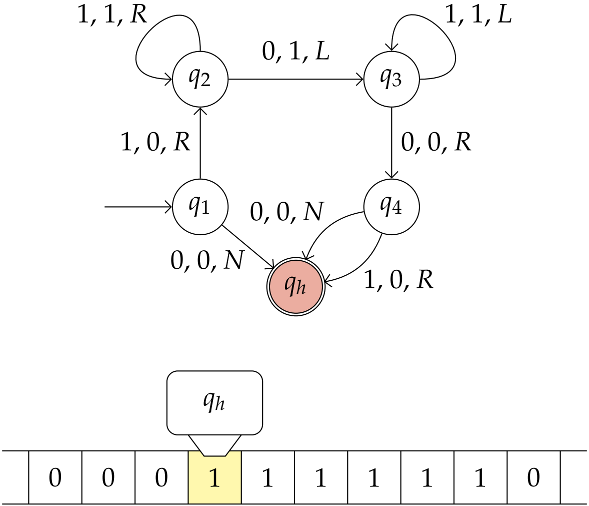


$t = 6$





$t = 7$



$t = 8$

This machine does:

- 3 strokes and 4 strokes (separated by a 0) $\mapsto$ 6 strokes

This machine does:

- 3 strokes and 4 strokes (separated by a 0) $\mapsto$ 6 strokes
- n strokes and m strokes (separated by a 0) $\mapsto$ $(n + m - 1)$ strokes

This machine does:

- 3 strokes and 4 strokes (separated by a 0) $\mapsto$ 6 strokes
- n strokes and m strokes (separated by a 0) $\mapsto$ $(n + m - 1)$ strokes

So we can interpret this machine as computing the function

- "the pair of n and m" $\mapsto n + m - 1$,

This machine does:

- 3 strokes and 4 strokes (separated by a 0) $\mapsto$ 6 strokes
- n strokes and m strokes (separated by a 0) $\mapsto$ $(n + m - 1)$ strokes

So we can interpret this machine as computing the function

- "the pair of n and m" $\mapsto n + m - 1$,

if we represent 1 with one stroke, 2 with two strokes, ...

This machine does:

- 3 strokes and 4 strokes (separated by a 0) $\mapsto$ 6 strokes
- n strokes and m strokes (separated by a 0) $\mapsto$ $(n + m - 1)$ strokes

So we can interpret this machine as computing the function

- "the pair of n and m" $\mapsto n + m - 1$,

if we represent 1 with one stroke, 2 with two strokes, ...

But that is not the only possible representation.

$$11101111000\cdots$$

can represent

a. the pair of 3 and 4, or $(3, 4)$ in the mathematician's notation;

This machine does:

- 3 strokes and 4 strokes (separated by a 0) $\mapsto$ 6 strokes
- n strokes and m strokes (separated by a 0) $\mapsto$ $(n + m - 1)$ strokes

So we can interpret this machine as computing the function

- "the pair of n and m" $\mapsto n + m - 1$,

if we represent 1 with one stroke, 2 with two strokes, ...

But that is not the only possible representation.

$$11101111000\cdots$$

can represent

a. the pair of 3 and 4, or $(3, 4)$ in the mathematician's notation;
b. $(2, 3)$, if we represent 0 with one stroke, 1 with two strokes, . . . ;

This machine does:

- 3 strokes and 4 strokes (separated by a 0) $\mapsto$ 6 strokes
- n strokes and m strokes (separated by a 0) $\mapsto$ $(n + m - 1)$ strokes

So we can interpret this machine as computing the function

- "the pair of n and m" $\mapsto n + m - 1$,

if we represent 1 with one stroke, 2 with two strokes, ...

But that is not the only possible representation.

$$11101111000\cdots$$

can represent

a. the pair of 3 and 4, or $(3, 4)$ in the mathematician's notation;
b. $(2, 3)$, if we represent 0 with one stroke, 1 with two strokes, . . . ;
c. 247 $(= 1 + 2 + 4 + 16 + 32 + 64 + 128)$ in the binary notation;

This machine does:

- 3 strokes and 4 strokes (separated by a 0) $\mapsto$ 6 strokes
- n strokes and m strokes (separated by a 0) $\mapsto$ $(n + m - 1)$ strokes

So we can interpret this machine as computing the function

- "the pair of n and m" $\mapsto n + m - 1$,

if we represent 1 with one stroke, 2 with two strokes, ...

But that is not the only possible representation.

$$11101111000\cdots$$

can represent

a. the pair of 3 and 4, or $(3, 4)$ in the mathematician's notation;
b. $(2, 3)$, if we represent 0 with one stroke, 1 with two strokes, . . . ;
c. 247 $(= 1 + 2 + 4 + 16 + 32 + 64 + 128)$ in the binary notation;

   etc., etc. . . .

(a) n strokes represent n.

(b) $n+1$ strokes represent n.

Using these representations, the machine that does

- n strokes and m strokes $\mapsto (n+m-1)$ strokes

can be interpreted as computing

(a) the function $(n, m) \mapsto n+m-1$ in the representation (a),

(a) n strokes represent n.

(b) $n+1$ strokes represent n.

Using these representations, the machine that does

- n strokes and m strokes $\mapsto (n+m-1)$ strokes

can be interpreted as computing

(a) the function $(n, m) \mapsto n + m - 1$ in the representation (a),

(b) the function $(n, m) \mapsto$?? in the representation (b).

ⓐ n strokes represent n.

ⓑ $n + 1$ strokes represent n.

Using these representations, the machine that does

- n strokes and m strokes $\mapsto (n + m - 1)$ strokes
- $(n + 1)$ strokes and $(m + 1)$ strokes $\mapsto (n + m + 1)$ strokes

can be interpreted as computing

ⓐ the function $(n, m) \mapsto n + m - 1$ in the representation ⓐ,

ⓑ the function $(n, m) \mapsto ??$ in the representation ⓑ.

(a) n strokes represent n.

(b) $n+1$ strokes represent n.

Using these representations, the machine that does

- n strokes and m strokes $\mapsto (n+m-1)$ strokes
- $(n+1)$ strokes and $(m+1)$ strokes $\mapsto (n+m+1)$ strokes

can be interpreted as computing

(a) the function $(n,m) \mapsto n+m-1$ in the representation (a),

(b) the function $(n,m) \mapsto n+m$ in the representation (b).

ⓐ n strokes represent n.

ⓑ $n + 1$ strokes represent n.

Using these representations, the machine that does

- n strokes and m strokes $\mapsto (n + m - 1)$ strokes
- $(n + 1)$ strokes and $(m + 1)$ strokes $\mapsto (n + m + 1)$ strokes

can be interpreted as computing

ⓐ the function $(n, m) \mapsto n + m - 1$ in the representation ⓐ,

ⓑ the function $(n, m) \mapsto n + m$ in the representation ⓑ.

NB. In this way, what function a Turing machine computes depends on coding! (A point which may be relevant to PHIL 223.)

Let's play a bit with
`https://turingmachine.io/`

Thus, Turing machines can be used to

1. compute functions, or
2. check whether the input satisfies certain properties.

You can kinda see that these are “mechanically done”, or that they are “effective procedures”, can’t you?

Thus, Turing machines can be used to

1. compute functions, or
2. check whether the input satisfies certain properties.

You can kinda see that these are “mechanically done”, or that they are “effective procedures”, can’t you?

We can regard 2 as a function from input to 1 (“Yes”) / 0 (“No”), so that 2 is a special case of 1.

Thus, Turing machines can be used to

1. compute functions, or
2. check whether the input satisfies certain properties.

You can kinda see that these are "mechanically done", or that they are "effective procedures", can't you?

We can regard 2 as a function from input to 1 ("Yes") / 0 ("No"), so that 2 is a special case of 1.

Definition. We say that a function is "Turing computable" to mean that there is a Turing machine that computes it (or that can be interpreted as computing it, in some representation of numbers on a tape).

Thus, Turing machines can be used to

1. compute functions, or
2. check whether the input satisfies certain properties.

You can kinda see that these are "mechanically done", or that they are "effective procedures", can't you?

We can regard 2 as a function from input to 1 ("Yes") / 0 ("No"), so that 2 is a special case of 1.

Definition. We say that a function is "Turing computable" to mean that there is a Turing machine that computes it (or that can be interpreted as computing it, in some representation of numbers on a tape).

It seems correct to say that anything Turing computable is computed mechanically or effectively. How about the opposite? We'll see . . . .

Turing Machines:
What is a Function?

We think of a **function** f as a box that takes a fixed number of numbers as inputs and outputs a fixed number of numbers.

In the mathematician's notation,

$$f : (n_1, \dots, n_k) \mapsto f(n_1, \dots, n_k).$$

We think of a **function** f as a box that takes a fixed number of numbers as inputs and outputs a fixed number of numbers.

In the mathematician's notation,

$$f : (n_1, \ldots, n_k) \mapsto f(n_1, \ldots, n_k).$$

For instance, addition is a function that takes a pair of numbers as inputs and outputs a number.

$$+ : (m, n) \mapsto m + n.$$

We think of a **function** f as a box that takes a fixed number of numbers as inputs and outputs a fixed number of numbers.

In the mathematician's notation,

$$f : (n_1, \ldots, n_k) \mapsto f(n_1, \ldots, n_k).$$

For instance, addition is a function that takes a pair of numbers as inputs and outputs a number.

$$+ : (m, n) \mapsto m + n.$$

Or
$$+ : \mathbb{N}^2 \to \mathbb{N}^1 :: (m, n) \mapsto m + n$$

to indicate the numbers of inputs (i.e. 2) and outputs (i.e. 1).

We think of a **function** f as a box that takes a fixed number of numbers as inputs and outputs a fixed number of numbers.

In the mathematician's notation,

$$f : (n_1, \ldots, n_k) \mapsto f(n_1, \ldots, n_k).$$

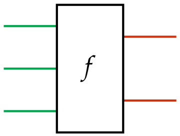
For instance, addition is a function that takes a pair of numbers as inputs and outputs a number.

$$+ : (m, n) \mapsto m + n.$$

Or $$+ : \mathbb{N}^2 \to \mathbb{N}^1 :: (m, n) \mapsto m + n$$

to indicate the numbers of inputs (i.e. 2) and outputs (i.e. 1).

In a picture, a box with input wires and output wires:



We think of a **function** f as a box that takes a fixed number of numbers as inputs and outputs a fixed number of numbers.

In the mathematician's notation,

$$f : (n_1, \ldots, n_k) \mapsto f(n_1, \ldots, n_k).$$

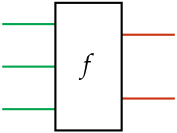
For instance, addition is a function that takes a pair of numbers as inputs and outputs a number.

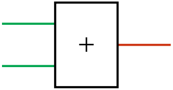
$$+ : (m, n) \mapsto m + n.$$

Or $$+ : \mathbb{N}^2 \longrightarrow \mathbb{N}^1 :: (m, n) \mapsto m + n$$

to indicate the numbers of inputs (i.e. 2) and outputs (i.e. 1).

In a picture, a box with input wires and output wires:





We think of a **function** f as a box that takes a fixed number of numbers as inputs and outputs a fixed number of numbers.

In the mathematician's notation,

$$f : (n_1, \ldots, n_k) \mapsto f(n_1, \ldots, n_k).$$

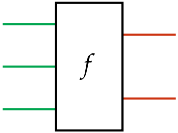
For instance, addition is a function that takes a pair of numbers as inputs and outputs a number.

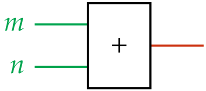
$$+ : (m, n) \mapsto m + n.$$

Or $$+ : \mathbb{N}^2 \longrightarrow \mathbb{N}^1 :: (m, n) \mapsto m + n$$

to indicate the numbers of inputs (i.e. 2) and outputs (i.e. 1).

In a picture, a box with input wires and output wires:





We think of a **function** f as a box that takes a fixed number of numbers as inputs and outputs a fixed number of numbers.

In the mathematician's notation,

$$f : (n_1, \ldots, n_k) \mapsto f(n_1, \ldots, n_k).$$

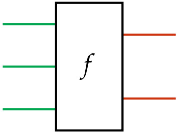
For instance, addition is a function that takes a pair of numbers as inputs and outputs a number.

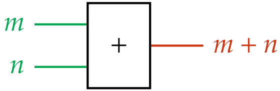
$$+ : (m, n) \mapsto m + n.$$

Or $$+ : \mathbb{N}^2 \longrightarrow \mathbb{N}^1 :: (m, n) \mapsto m + n$$

to indicate the numbers of inputs (i.e. 2) and outputs (i.e. 1).

In a picture, a box with input wires and output wires:





According to the mathematician's (usual) definition, functions f must satisfy two conditions:

1. Once you feed in an input (a combination of numbers), the output must be unique. I.e., there cannot be two different outputs for the same input.
2. Once you feed in an input, f must output something. I.e., $f(n_1, \ldots, n_k)$ is always defined for all possible inputs $(n_1, \ldots, n_k)$.

According to the mathematician's (usual) definition, functions f must satisfy two conditions:

1. Once you feed in an input (a combination of numbers), the output must be unique. I.e., there cannot be two different outputs for the same input.
2. Once you feed in an input, f must output something. I.e., $f(n_1, \ldots, n_k)$ is always defined for all possible inputs $(n_1, \ldots, n_k)$.

For CS, the requirement 2 is too strong (explained shortly). Hence, by a function, we mean a box satisfying 1; it may or may not satisfy 2.

According to the mathematician's (usual) definition, functions f must satisfy two conditions:

1. Once you feed in an input (a combination of numbers), the output must be unique. I.e., there cannot be two different outputs for the same input.
2. Once you feed in an input, f must output something. I.e., $f(n_1, \ldots, n_k)$ is always defined for all possible inputs $(n_1, \ldots, n_k)$.

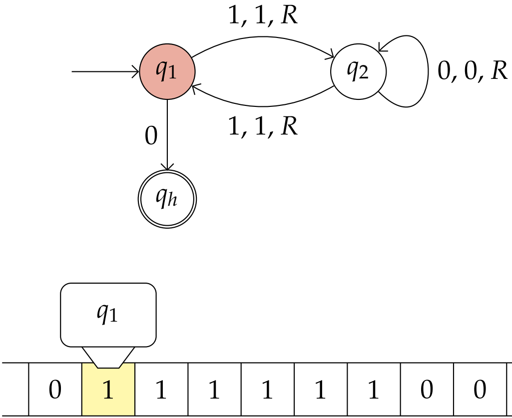
For CS, the requirement 2 is too strong (explained shortly). Hence, by a function, we mean a box satisfying 1; it may or may not satisfy 2.

In technical terms:

Definition. A "partial function" f is a mapping of numbers to numbers s.th., although the value $f(n_1, \ldots, n_k)$ may be undefined, if it is defined then it is unique.

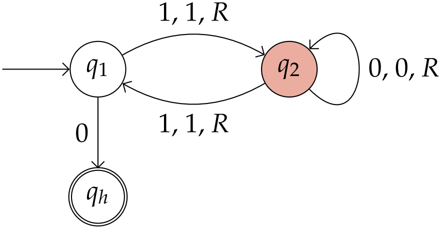
We say a function is "total" to mean that its values are always defined.

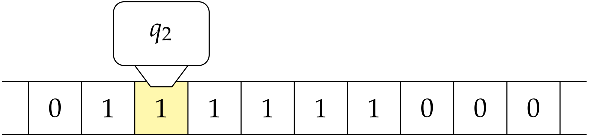
Consider the following Turing machine.



$t = 0$

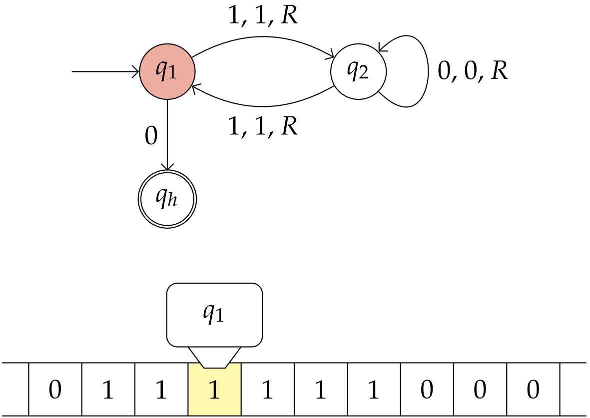
Consider the following Turing machine.





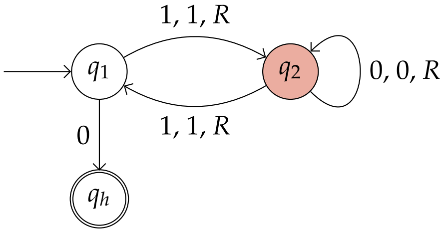
$t = 1$

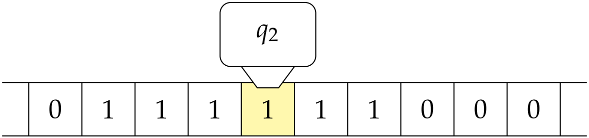
Consider the following Turing machine.



$t = 2$

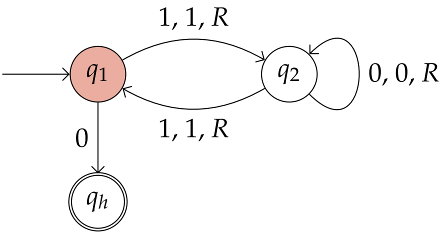
Consider the following Turing machine.

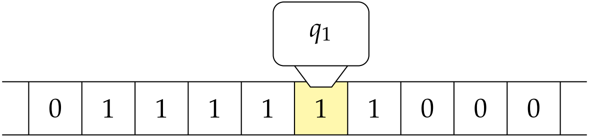




$t = 3$

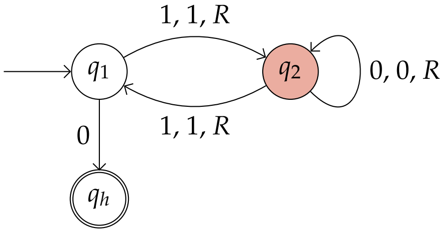
Consider the following Turing machine.

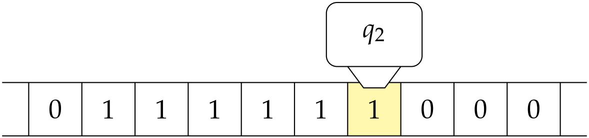




$t = 4$

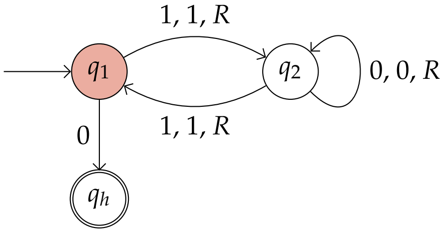
Consider the following Turing machine.

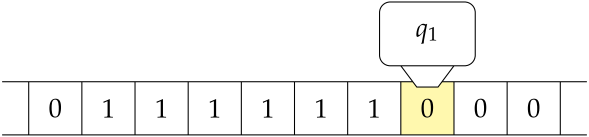




$t = 5$

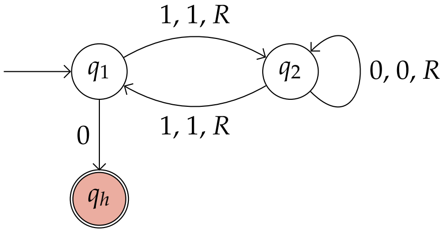
Consider the following Turing machine.

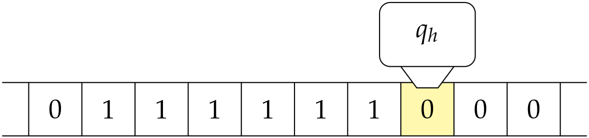




$t = 6$

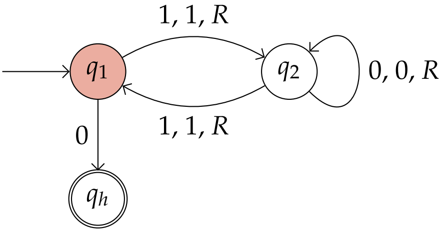
Consider the following Turing machine.

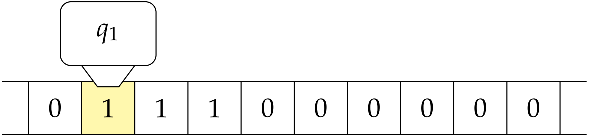




$t = 7$

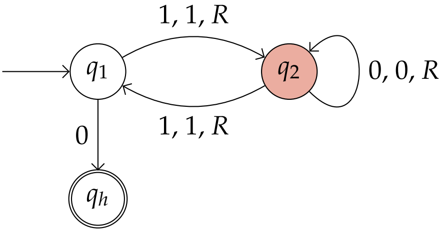
Consider the following Turing machine.

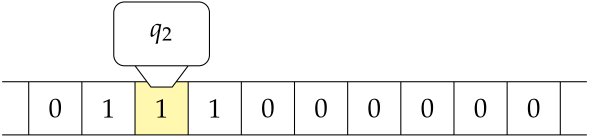




$t = 0$

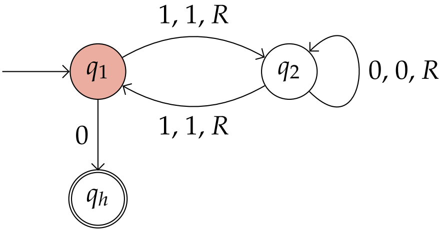
Consider the following Turing machine.

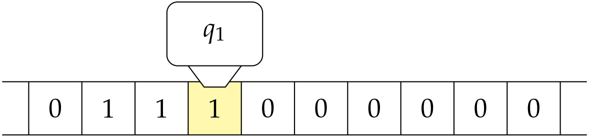




$t = 1$

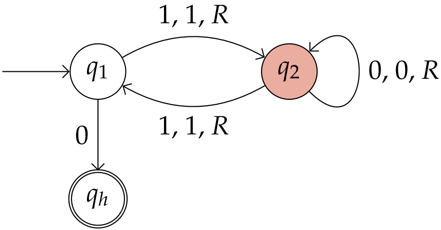
Consider the following Turing machine.

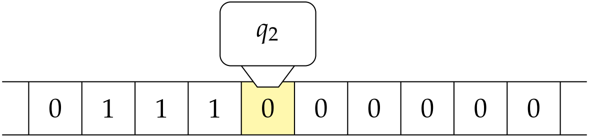




$t = 2$

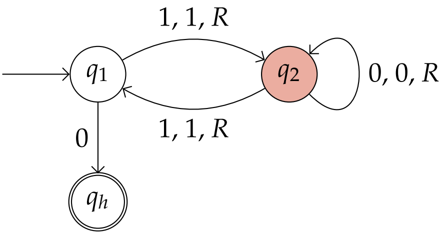
Consider the following Turing machine.

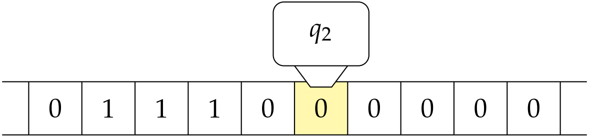




$t = 3$

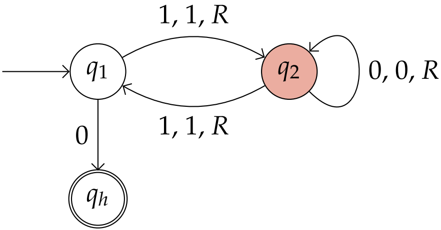
Consider the following Turing machine.

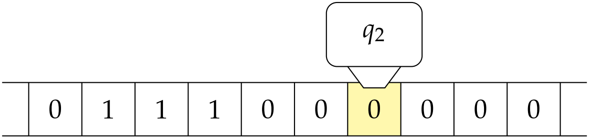




$t = 4$

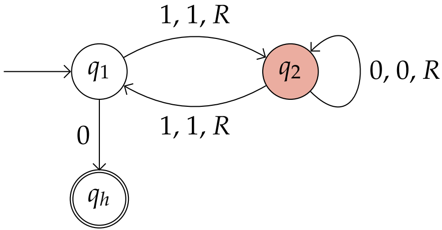
Consider the following Turing machine.

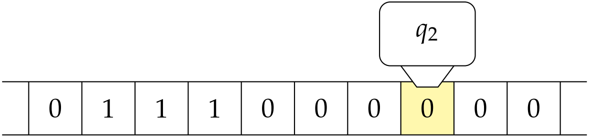




$t = 5$

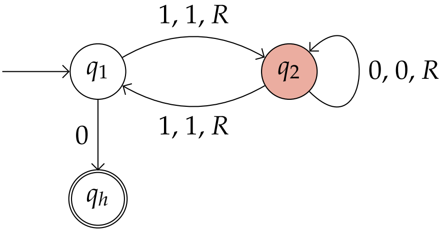
Consider the following Turing machine.

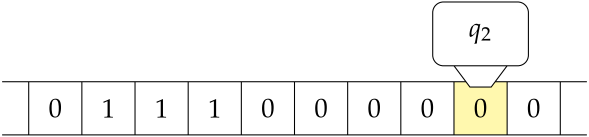




$t = 6$

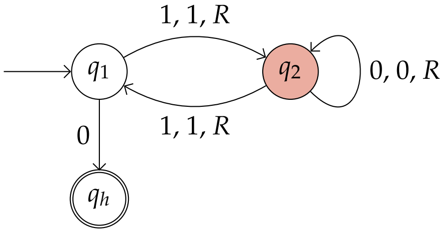
Consider the following Turing machine.

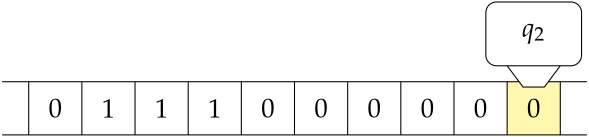




$t = 7$

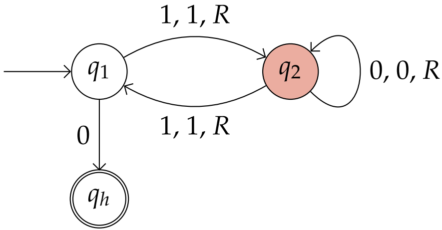
Consider the following Turing machine.

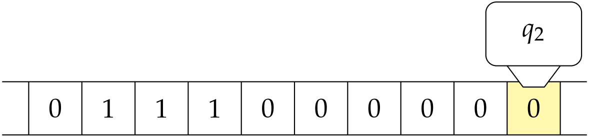




$t = 8$

Consider the following Turing machine.





$t = 8$

Some Turing machines, depending on the input, may never halt!