A 14 tile aperiodic Wang tile set

A very different algebraic method of constructing aperiodic tile sets yields a set with only 14 tiles.

(But even smaller aperiodic sets exist: E. Jeandel and M. Rao have an aperiodic Wang tile set that contains just 11 tiles, and they proved that 11 is the smallest possible size.)

Undecidable problems concerning tiles

Question: How can one determine if a given finite set of Wang prototiles admits a tiling ?

This is an **algorithmic** question (looking for an algorithm to solve the problem).

More specifically, it is a **decision problem**: the desired algorithm

- takes an input, called an **instance** of the problem. (. . . **given** finite set of Wang tiles. . .)
- should return the correct **"yes"** or **"no"** answer. (So "yes" if and only of the tile set can tile the plane.)

It turns out that such an algorithm **does not exist**: any candidate algorithm either gives a wrong answer for some tile sets, or gives no answer at all for some instances. We say the decision problem is **undecidable**.

An **algorithm** can be formally defined in various, equivalent ways.

For our purposes it is sufficient to understand a (decision) algorithm to be a **computer program** that takes some input and returns a "yes" or a "no" answer on each input. We say that the algorithm **solves** a decision problem if the algorithm returns the correct yes/no -answer on every instance of the problem.

If such an algorithm exists then the decision problem is called **decidable** and if no such algorithm exists then the problem is **undecidable**.

A **semi-algorithm** is a weaker concept than an algorithm: it is a computer program that must halt and return "yes" if the input is a positive instance of the problem, but it is allowed to run forever, without ever halting, on negative input instances. (It may never return a wrong answer.)

If a decision problem has a semi-algorithm then it is called **semi-decidable**. Clearly every decidable problem is also semi-decidable as an algorithm is also a semi-algorithm.

Example. The following decision problem is semi-decidable: Does a given Wang tile set admit any periodic tiling of the plane?

Periodic tiling problem

Instance: A finite set $\mathcal{T}$ of Wang tiles

Positive instance: $\mathcal{T}$ admits a periodic tiling of the plane

A **semi-algorithm** for the Periodic tiling problem:

For $n = 1, 2, \ldots$ do the following: Try all assignments

$$f : \{1, \ldots n\} \times \{1, \ldots, n\} \longrightarrow \mathcal{T}$$

of tiles to the $n \times n$ square. For each f check whether

- the square is correctly tiled (no mismatching neighboring tiles),
- the sequence of colors on the left border is the same as on the right border, and the sequence of colors on the top border is the same as on the bottom border.

If an assignment f is found that satisfies both conditions then halt and return answer "yes".

The **complement** problem of a decision problem is the problem where the "yes" and "no" instances have been swapped.

Note that the complement of a decidable problem is also decidable: the same algorithm works, just swap the answer

$$\text{"yes"} \longleftrightarrow \text{"no"}$$

Example. The complement of **Periodic tiling problem** is the problem whose positive instances are the tile sets that do **not** admit any periodic tiling.

If the complement of problem P is semi-decidable we also may say that the negative instances of P are semi-decidable. Semi-decidability of P means that the positive instances are semi-decidable.

Example. The **complement** of the following decision problem is semi-decidable:

Tiling problem
Instance: A finite set $\mathcal{T}$ of Wang tiles
Positive instance: $\mathcal{T}$ admits a tiling of the plane

A semi-algorithm for the negative instances (tiles that do not tile the plane):

For $n = 1, 2, \ldots$ do the following: Try all assignments

$$f : \{1, \ldots n\} \times \{1, \ldots, n\} \longrightarrow \mathcal{T}$$

of tiles to the $n \times n$ square. If for some n none of the assignments is a valid tiling of the $n \times n$ square, return "no" as there is no tiling of the plane.

Theorem. If problem P and the complement of P are both semi-decidable, then P is decidable.

Proof. Run the semi-algorithms for P and its complement in parallel. Eventually one of them gives an answer.

Positive instances of **Periodic tiling problem** and negative instances of **Tiling problem** are semi-decidable.

If there were no aperiodic tile sets, **Periodic tiling problem** and **Tiling problem** would be the same problem. The problem would then be decidable!

(But there are aperiodic tile sets, and these fall in between the two semi-algorithms: on aperiodic tile set both of our semi-algorithms do not halt but continue forever on larger-and-larger $n \times n$ squares.)

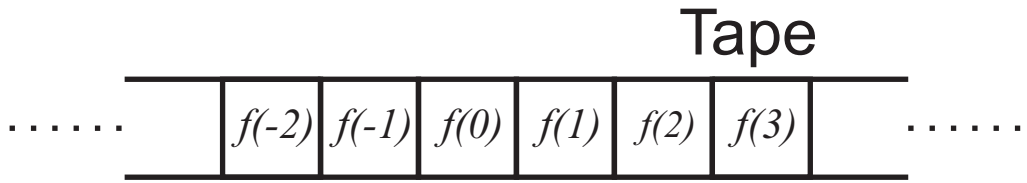
Turing machines

In computation theory Turing machines (TM) are used as a possible formalism for algorithms. Here, we use them as instances of a basic undecidable problem, the **halting problem** of TM. This is a seed problem that we reduce to various tiling problems to show that they are undecidable as well.

A Turing machine is an object with:

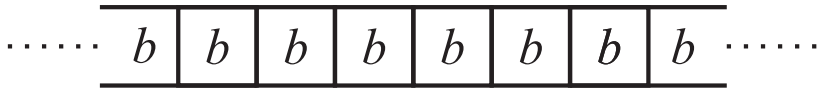
- a bi-infinite **tape** of cells indexed by $\mathbb{Z}$,
- a finite set Γ, the **tape alphabet**, of symbols that are written in the tape cells.

So the **content** of the tape is then a function $f : \mathbb{Z} \longrightarrow \Gamma$ where $f(i)$ is the symbol at location i. The set of possible tape contents is $\Gamma^{\mathbb{Z}}$.

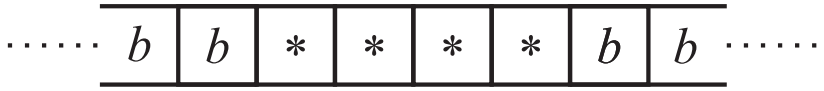


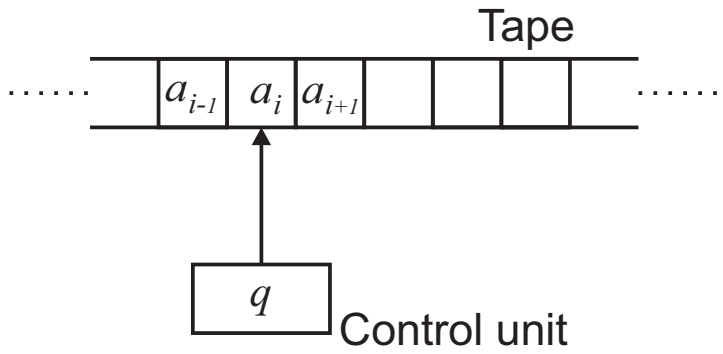
One element $b \in \Gamma$ is specified as the **blank symbol**.

- A tape content is the **blank tape** if every cell contains b.



- A tape content is **finite** if the number of cells with a non-b symbol is finite.





There is also a **control unit**, or a processor, that has access to one tape cell at any time. The unit is in some state q that is an element of a finite **state set** S.

TM operates at discrete time steps: depending on the

- current state, and
- the tape symbol currently in the accessed tape cell,

the TM may

- change its state,
- replace the tape symbol in the accesses tape cell, and
- change the accessed cell by moving the control unit one cell position to the left or right on the tape.

The action is specified by the **transition function**

$$\delta : S \times \Gamma \longrightarrow S \times \Gamma \times \{L, R\}.$$

The interpretation of

$$\delta(q, x) = (p, y, d)$$

is that if

- the current state is q, and
- the tape symbol at the current location i is x

then the machine

- changes the state into p,
- replaces x by y on the tape, and
- moves one position left (to position $i - 1$) or right (to position $i + 1$) on the tape depending on whether $d = L$ or $d = R$.

A **configuration** of the TM is an element of the set

$$S \times \mathbb{Z} \times \Gamma^{\mathbb{Z}}.$$

Configuration (q, i, f) specifies

- the current state $q \in S$,
- the current position $i \in \mathbb{Z}$ on the tape, and
- the current tape content $f : \mathbb{Z} \longrightarrow \Gamma$.

We can now define one move of the machine formally: Configuration (q, i, f) is transformed in one move into the configuration (p, j, g) if

- $\delta(q, f(i)) = (p, y, d)$,
- $g(i) = y$, $g(k) = f(k)$ for all $k \neq i$,
- $j = i + 1$ if $d = R$ and $j = i - 1$ if $d = L$.

We denote this move by

$$(q, i, f) \vdash (p, j, g).$$

In the beginning of the computation the Turing machine is in one specific state $s_0 \in S$ called the **initial state**, and another state $s_h \in S$ is specified as the **halting state**.

The Turing machine halts when the control unit enters state s_h.

In the beginning of the computation the Turing machine is in one specific state $s_0 \in S$ called the **initial state**, and another state $s_h \in S$ is specified as the **halting state**.

The Turing machine halts when the control unit enters state s_h.

The Turing machine can be understood as a dynamical system where the transformation $\vdash$ is applied repeatedly starting from the initial configuration $(s_0, 0, f)$ where f is the initial tape content (usually blank or at least finite) until (if ever) the machine reaches and halts in some configuration (s_h, i, g), where i and g can be arbitrary.

Note that if the initial tape content is finite then it remains finite at all times. This means that configurations (s, i, f) during the operation have finite representations.

So, to specify a Turing machine one needs to provide six items. We say that a Turing machine is a six-tuple

$$M = (S, \Gamma, \delta, s_0, s_h, b)$$

where S and Γ are finite sets, $s_0, s_h \in S$ and $b \in \Gamma$ are elements of those sets, and $\delta : S \times \Gamma \longrightarrow S \times \Gamma \times \{L, R\}$ is a function.

Example. Consider the following TM

$$M = (\{s, t, h\}, \{a, b\}, \delta, s, h, b)$$

where

$$\begin{aligned} \delta(s, a) &= (t, a, L) \\ \delta(s, b) &= (t, a, R) \\ \delta(t, a) &= (h, a, L) \\ \delta(t, b) &= (s, a, L) \end{aligned}$$

(and the values of $\delta(h, \dots)$ do not matter as h is the halting state.)

The operation of M from the initial blank tape:

Two basic decision problems concerning Turing machines:

Halting from blank tape
Instance: A TM $M = (S, \Gamma, \delta, s_0, s_h, b)$
Positive instance: M such that $(s_0, 0, f) \vdash \cdots \vdash (s_h, i, g)$ for some and $i \in \mathbb{Z}$ and $g \in \Gamma^{\mathbb{Z}}$, where f is the blank tape.

The second problem is for a fixed TM $M = (S, \Gamma, \delta, s_0, s_h, b)$:

Halting problem of TM $M = (S, \Gamma, \delta, s_0, s_h, b)$
Instance: A finite tape content $f \in \Gamma^{\mathbb{Z}}$
Positive instance: f such that $(s_0, 0, f) \vdash \cdots \vdash (s_h, i, g)$ for some and $i \in \mathbb{Z}$ and $g \in \Gamma^{\mathbb{Z}}$

The first problem is undecidable. For some M the second problem is decidable, but there are also such M (universal Turing machines) that the second problem is undecidable:

Theorem [Turing 1936]. There exists a TM M such that **Halting problem of TM M** is undecidable.

Once we know one undecidable problem we use it as a seed to prove other problems undecidable, using **reductions**.

Turing reduction. Let P be a decision problem that we know to be undecidable. Let Q be another decision problem. This is how we prove that Q is undecidable:

- Make the assumption that there exists an algorithm A that solves Q.
- Design an algorithm that solves P, using the hypothetical A as a subroutine.

Now since the algorithm to solve P does not exist, the hypothetical A cannot exist either. Thus Q is undecidable.

Example. Let us prove that **Halting from blank tape** is undecidable. Suppose the contrary: There exists algorithm A that solves it. Then we can solve **Halting problem of TM** M by the following algorithm (for any fixed machine M):

The input is a finite tape content f to M. We construct a Turing machine M' that has all the states and transitions of M and some new ones so that M' operates as follows when started in its initial state:

- writes to the initially empty tape the new content f,
- returns to the position 0,
- enters the initial state of M.

The algorithm now calls the hypothetical algorithm A with input M', and returns whatever answer A returns.

This algorithm correctly solves **Halting problem of TM** M because

M' halts on empty initial tape $\iff$ M halts on input f.

The reduction we made is an example of a **many-one** reduction (a restricted kind of Turing reduction). All our reductions will be of this type:

Let P be a decision problem that we know to be undecidable. Let Q be another decision problem. This is how we prove that Q is undecidable:

- Show how for any given instance x of problem P we can effectively (i.e. algorithmically) construct an equivalent instance y of Q.

(Equivalent means that x is a positive instance of P if and only if y is a positive instance of Q.)

Now any algorithm A to solve Q can be used to solve P by first converting the input x to an equivalent instance y of Q, and calling algorithm A with input y. Thus A cannot exist.

In our example: we constructed for a given finite initial tape content f a new TM M' that halts on the empty initial tape if and only if M halts on initial tape content f.