Undecidability: More Reductions

Thank you to viewers like you

Announcements about Midterm

- Topics on midterm:

  Beginning of course through Monday 2/19 lecture (not today's lecture)

  ⇒ Includes Turing reductions, but not the type you'll learn today where you construct another machine

- Practice midterms from previous terms have been released
- You may bring one double-sided 8.5 x 11 study sheet, that you prepare
- Midterm review session tomorrow 2/22 6-8pm LMBE 1130 with Daphne

  Topic: Turing Reductions and Dynamic Programming

- The week after break:
  - Monday 3/4 lecture: midterm review
  - No lecture on Wednesday 3/6
  - Midterm is Wednesday 3/6: 7-9pm

Other Admin

There will be extra office hours on Thursday (see website) since the HW is due Thursday

Reminder: Filling out the course evaluations is 1% of your grade, which is otherwise covered by the final exam

So far we've shown these languages are undecidable:

- $L_{BARBER} = \{\langle M \rangle : M \text{ does not accept } \langle M \rangle\}$
- $L_{ACC} = \{(\langle M \rangle, x) : M \text{ accepts } x\}$
- $L_{HALT} = \{(\langle M \rangle, x) : M \text{ halts on input } x\}$

We did this using two proof techniques:

1. **Diagonalization**/paradox

1. **Reduction** from a problem we already knew was undecidable

Turing Reduction from A to B (denoted $A \leq_T B$):

"We can use a black-box decider for B
as a subroutine to decide A."

What it implies:

1. If B is decidable then A is decidable.
2. Contrapositive: If A is undecidable then B is undecidable.

You can even invoke M_B multiple times inside of M_A if you want

input to M_A

M_A: Decider for A

1. do some stuff
2. give input to M_B

M_B: Decider for B

3. get output from M_B
4. do some more stuff

output of M_A

"Problem B is at least as hard as Problem A"

Review: Reduction from L_{ACC} to L_{HALT}

A B

We need to implement:

M_{ACC} takes two inputs: ⟨M⟩, x

M accepts x ⇒ M_{ACC} accepts

M loops or rejects x ⇒ M_{ACC} rejects

Suppose we have:

M_{HALT} takes two inputs: ⟨M⟩, x

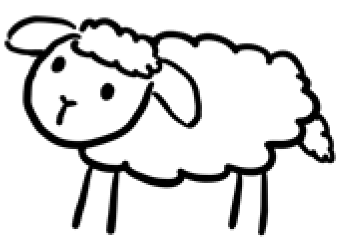
M accepts or rejects x ⇒ M_{HALT} accepts

M loops on input x ⇒ M_{HALT} rejects

We need to specify the pseudocode:

```
MACC(⟨M⟩, x):
    Run MHALT(<M>, x)
        If it rejects: reject
    Otherwise, run M(x)
        If it accepts: accept
        If it rejects: reject
```

We are allowed to use M_{HALT}(⟨M⟩, x) as a subroutine, with the inputs of our choice

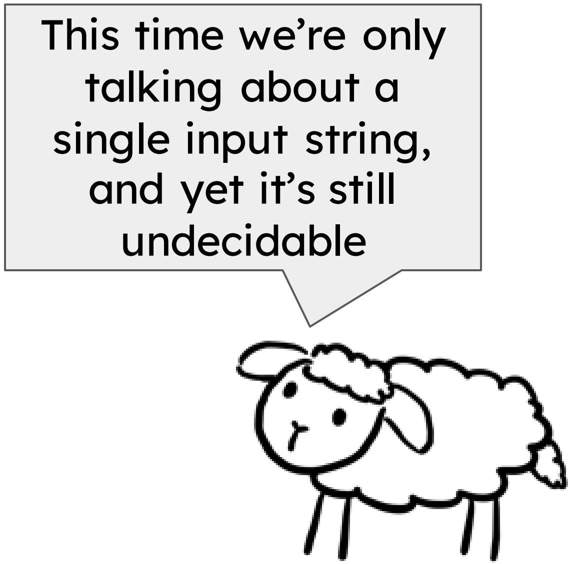


Another Undecidable Language:
ε-Halting Problem

Input: Turing Machine **M**

Output: Does **M** halt when given input **ε**?

Language: $L_{\varepsilon\text{-HALT}} = \{\langle M \rangle : M \text{ halts on input } \varepsilon\}$



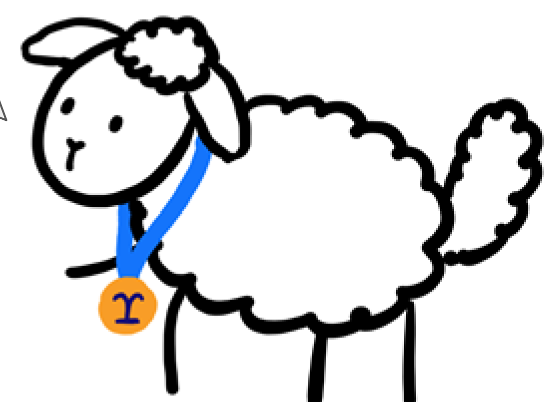
Here's a reduction from $L_{\varepsilon\text{-HALT}}$ to L_{HALT}, showing $L_{\varepsilon\text{-HALT}}$ is undecidable!

$M_{\varepsilon\text{-HALT}}(\langle M \rangle)$:

Run $M_{\text{HALT}}(\langle M \rangle, \varepsilon)$

If it accepts: accept

If it rejects: reject



Something is off…

Reduction from L_{HALT} to $L_{\varepsilon\text{-HALT}}$ (i.e. $L_{\text{HALT}} \leq_T L_{\varepsilon\text{-HALT}}$)

We need to implement:

M_{HALT} takes two inputs: $\langle M \rangle$, x

M halts on input $x \Rightarrow M_{\text{HALT}}$ accepts

M loops on input $x \Rightarrow M_{\text{HALT}}$ rejects

Suppose we have:

$M_{\varepsilon\text{-HALT}}$ takes one input: $\langle M \rangle$

M halts on input $\varepsilon \Rightarrow M_{\text{ACC}}$ accepts

M loops on input $\varepsilon \Rightarrow M_{\text{ACC}}$ rejects

We need to specify the pseudocode:

$M_{\text{HALT}}(\langle M \rangle, x)$:

$M_{\varepsilon\text{-HALT}}(\langle M_x \rangle)$

M_x : TM that runs $M(x)$

We are allowed to use $M_{\varepsilon\text{-HALT}}(\langle M \rangle)$ as a subroutine, with the input of our choice

Reduction from L_{HALT} to $L_{\varepsilon\text{-}HALT}$ (i.e. $L_{HALT} \leq_T L_{\varepsilon\text{-}HALT}$)

We need to implement:

M_{HALT} takes two inputs: ⟨M⟩, x

M halts on input x ⇒ M_{HALT} accepts

M loops on input x ⇒ M_{HALT} rejects

Suppose we have:

$M_{\varepsilon\text{-}HALT}$ takes one input: ⟨M⟩

M halts on input ε ⇒ M_{ACC} accepts

M loops on input ε ⇒ M_{ACC} rejects

⟨M_x⟩

We need to specify the pseudocode:

M_{HALT}(⟨M⟩, x):

"
M_x(w):
  Run M(x) and answer as M "

Let M_x be a TM that ignores its input and runs M(x)

Run $M_{\varepsilon\text{-}HALT}$(⟨$M_x$⟩) and answer as $M_{\varepsilon\text{-}HALT}$

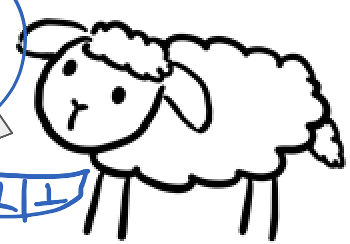
M halts (doesn't) on x → $M_x(w)$ halts (loops) for all w, including w = ε

→ $M_{\varepsilon\text{-}HALT}(M_x)$ accepts (rejects)

→ $M_{HALT}(\langle M\rangle, x)$ accepts (reject)

Note: We didn't actually run M_x, we just constructed it

Key idea: Since our subroutine only takes one input, encode x (x+ε) into the definition of the TM M_x



Another Undecidable Language:
Membership Oracle

Input: Turing Machine **M**

Output: Does **M** accept when given input **376**?

Language: $L_{376} = \{\langle M \rangle : 376 \in L(M)\}$

L_{376} is to L_{ACC}
as
$L_{\varepsilon\text{-HALT}}$ is to L_{HALT}

Reduction from L_{Acc} to L_{376} (i.e. $L_{Acc} \le_T L_{376}$)

We need to implement:

M_{Acc} takes two inputs: **⟨M⟩, x**

M accepts **x** ⇒ **M_{Acc}** accepts

M doesn't accept **x** ⇒ **M_{Acc}** rejects

Suppose we have:

M_{376} takes one input: **⟨M⟩**

M accepts **376** ⇒ **M_{376}** accepts

M doesn't accept **376** ⇒ **M_{376}** rejects

We need to specify the pseudocode:

~~M_{Acc}(⟨M⟩, x):~~

Define M_x as before

$M_{376}(\langle M_x \rangle)$ and answer as M_{376}

$M_x(w)$:

Run M(x) ~~and answer as M~~

if M(x) accepts: accept

if M(x) rejects: reject

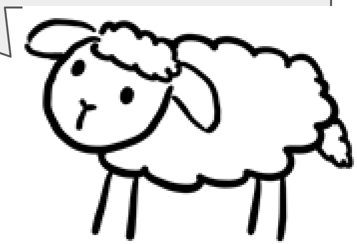
doesn't

M accepts x → $M_x(w)$ doesn't accepts for all w incl w=376

→ $M_{376}(\langle M_x \rangle)$ reject accept

→ $M_{Acc}(\langle M \rangle, x)$ reject accept

We are allowed to use M_{376}(⟨M⟩) as a subroutine, with the input of our choice



Reduction from L_{ACC} to L_{376} (i.e. $L_{ACC} \leq_T L_{376}$)

We need to implement:

M_{ACC} takes two inputs: $\langle M \rangle$, x

M accepts $x \Rightarrow M_{ACC}$ accepts

M doesn't accept $x \Rightarrow M_{ACC}$ rejects

Suppose we have:

M_{376} takes one input: $\langle M \rangle$

M accepts $376 \Rightarrow M_{376}$ accepts

M doesn't accept $376 \Rightarrow M_{376}$ rejects

We need to specify the pseudocode:

$M_{ACC}(\langle M \rangle, x)$:

Define M_x as before

$M_x(w)$:

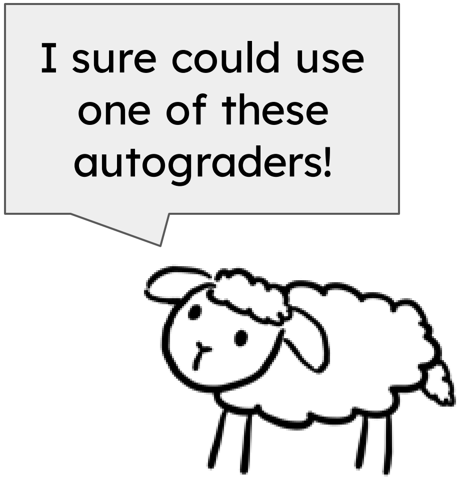
Run $M(x)$ and answer as M

Another Undecidable Language:
The Autograder Problem

Input: Two Turing Machines M_1, M_2

Output: Do M_1 and M_2 accept the same set of inputs?
I.e. is $L(M_1) = L(M_2)$?

Language: $L_{EQ} = \{\langle M_1 \rangle, \langle M_2 \rangle : L(M_1) = L(M_2)\}$



Reduction from L_{ACC} to L_{EQ} (i.e. $L_{ACC} \leq_T L_{EQ}$)

We need to implement:

M_{ACC} takes two inputs: $\langle M \rangle$, x

M accepts $x \Rightarrow M_{ACC}$ accepts

M doesn't accept $x \Rightarrow M_{ACC}$ rejects

Suppose we have:

M_{EQ} takes two inputs: $\langle M_1 \rangle$, $\langle M_2 \rangle$

$L(M_1) = L(M_2) \Rightarrow M_{EQ}$ accepts

$L(M_1) \neq L(M_2) \Rightarrow M_{EQ}$ rejects

We need to specify the pseudocode:

$M_{ACC}(\langle M \rangle, x)$:

> $M_x(w)$:
> Run $M(x)$ and answer as M

M accepts (doesn't) $x \to M_x(w)$ accepts (doesn't) for all w

Define M_x as before

$L(M_2) = \Sigma^*$

Let M_2 be TM that accepts every input

Run $M_{EQ}(\langle M_x \rangle, \langle M_2 \rangle)$ and answer as M_{EQ}

We are allowed to use $M_{EQ}(\langle M_1 \rangle, \langle M_2 \rangle)$ as a subroutine, with the inputs of our choice

$\to M_{EQ}(\langle M_x \rangle, \langle M_2 \rangle)$ accepts / reject

$\to M_{ACC}(\langle M \rangle, x)$ accepts / reject

$$L(M_x) = \begin{cases} \varnothing & M \text{ loops or rejects } x \\ \Sigma^* & M \text{ accepts } x \end{cases}$$

Question: Do all undecidable problems involve Turing machines?

Answer: **No!**

Question: Can the definition of a Turing machine be useful in proving undecidability?

Answer: **Yes!**

Another Undecidable Language:
Tiling the Plane

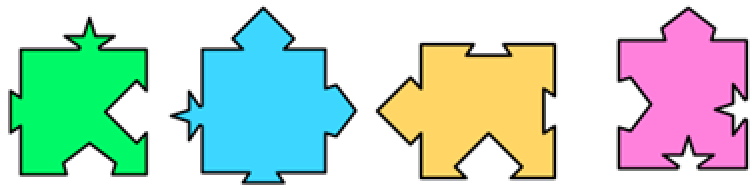
Input: Finite set **T** of 2-dimensional shapes ("tiles").

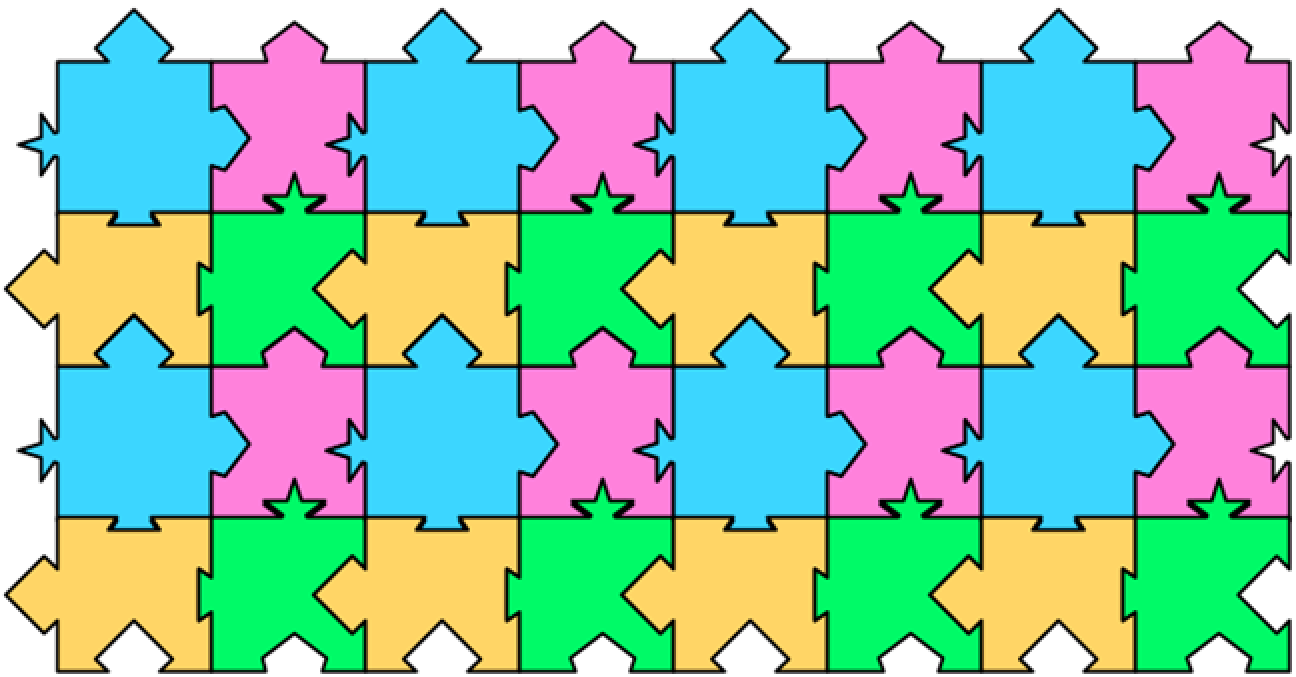
Output: Can we tile the plane using shapes from **T**? (We can use any rotation of each shape arbitrarily many times. No overlaps or gaps allowed.)

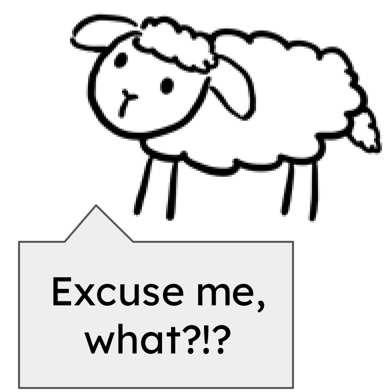
Another Undecidable Language:

Tiling the Plane

T =







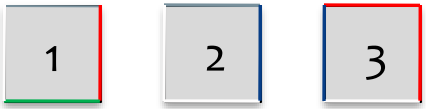
If you can solve the tiling problem then you can solve the halting problem!

We will focus on this related problem:

Wang Tilings (1966)

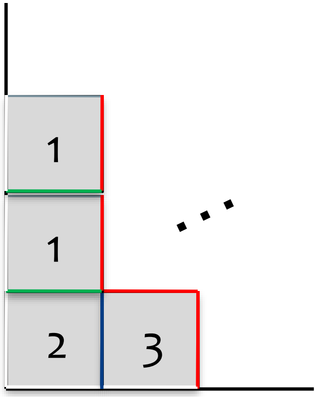
Hao Wang

Input: Finite set **T** of square tiles where each side of each square has a color.



Output: Can we tile the positive quadrant of the plane using tiles from **T** such that:

- Two squares are adjacent only if their colors match
- The boundary of the quadrant is colored white
- Squares cannot be rotated



Reduction from ε-Halting to Tiling

Suppose we have a black-box decider M_{TILE} for the Tiling Problem.

We will use it to construct pseudocode for $M_{\varepsilon\text{-}HALT}(\langle M \rangle)$:

$M_{\varepsilon\text{-}HALT}(\langle M \rangle)$

Run M_{TILE}(**tiles that we define!**) and answer the opposite of M_{TILE}

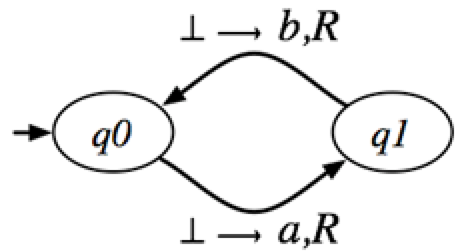
To prove correctness we want to show:

we can tile the (positive quadrant of the) plane ⇔ $M(\varepsilon)$ loops

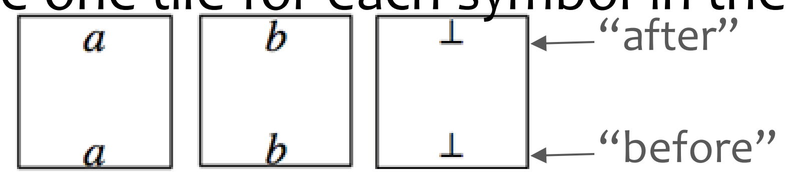
Ideas for Reduction from ε-Halting to Tiling
(not full proof)

Goal: Given **M**, construct tiles so that **we can tile the plane** ⇔ **M(ε) loops**

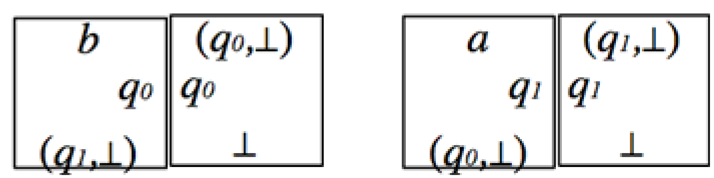
Running example of M:



1. Make one tile for each symbol in the tape alphabet of **M**:

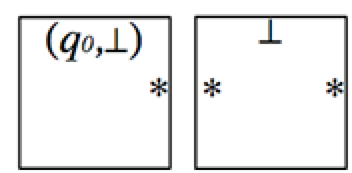


2. Make some tiles for each transition:



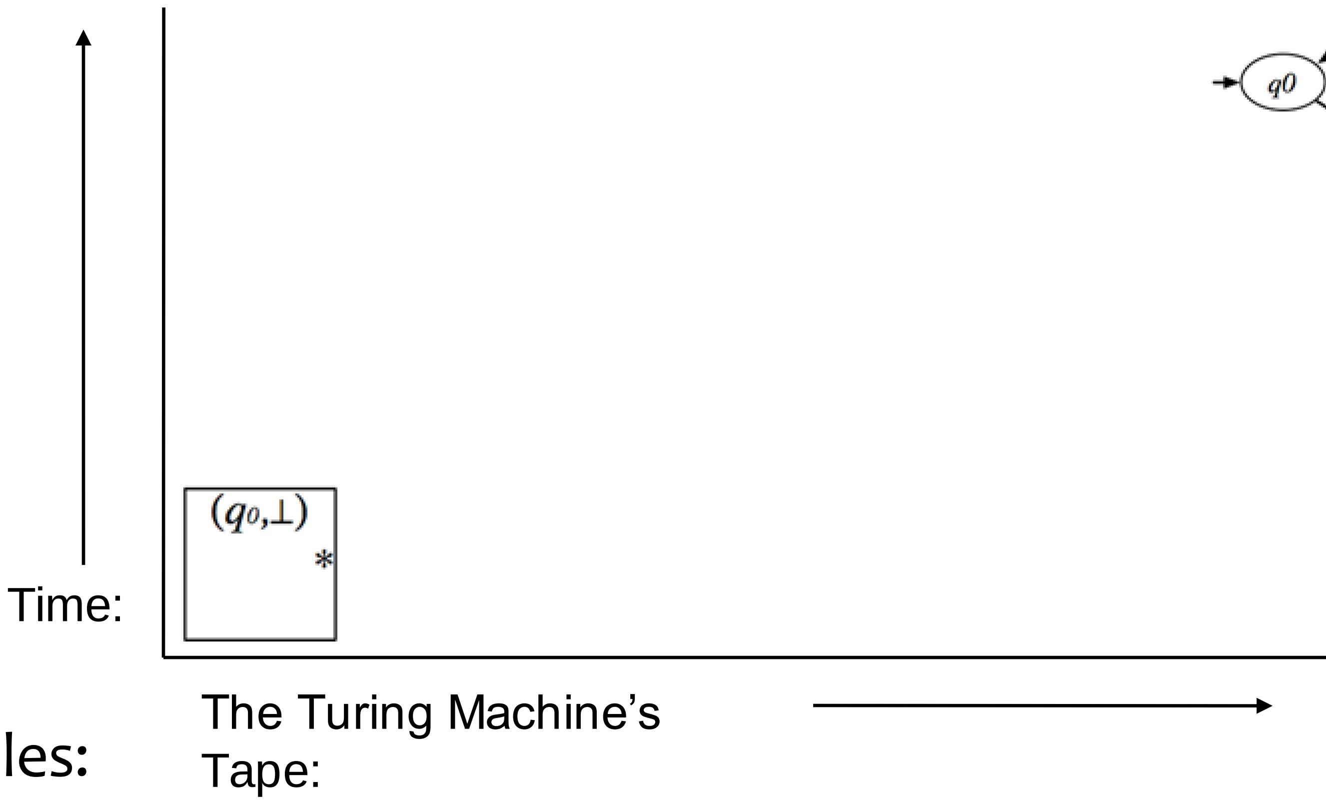
See online notes to construct tiles for an arbitrary TM

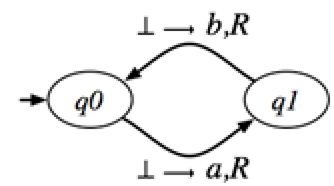
3. Make two special tiles for the start state and symbol:



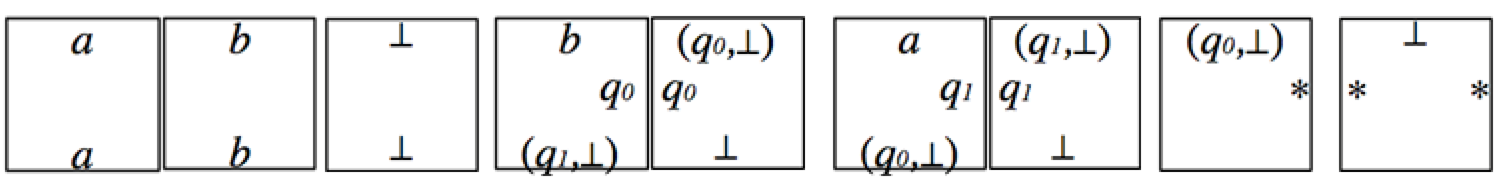
Claim: We can tile the plane $\Leftrightarrow$ **M(ε)** loops forever

Only one tile is white on both corner edges



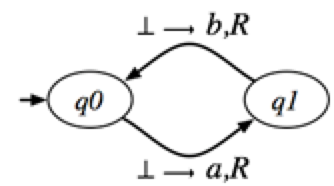


Set of Tiles:



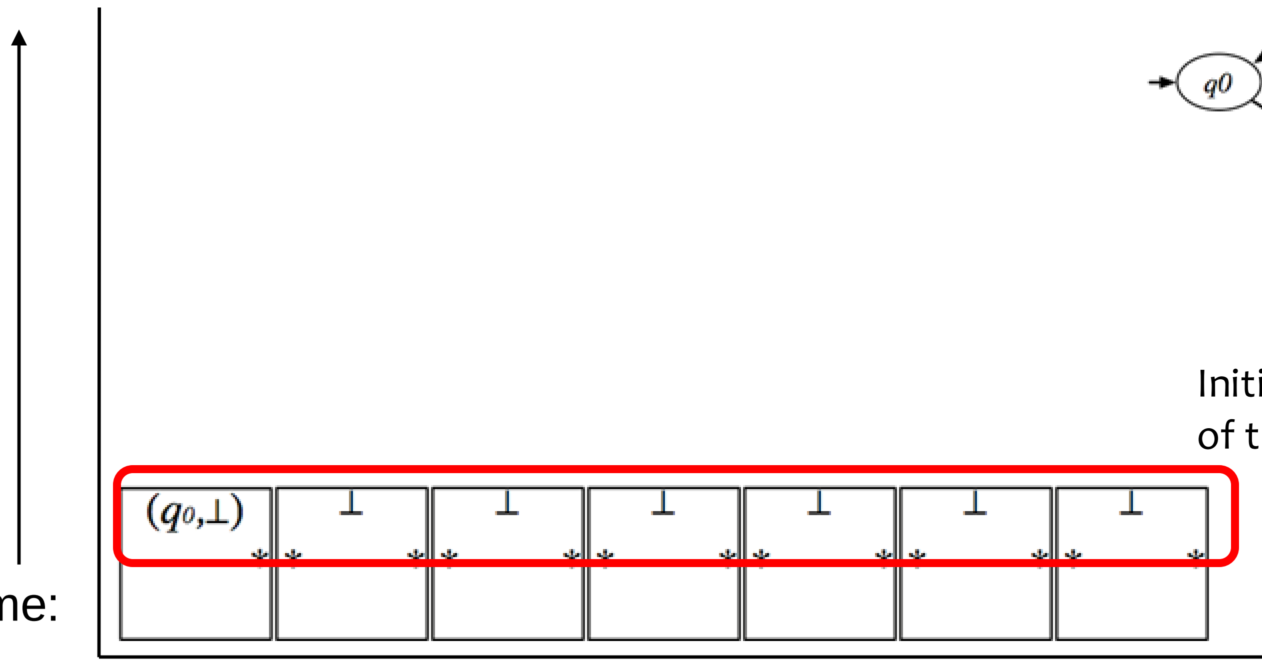
Claim: We can tile the plane ⇔ **M(ε)** loops forever

Only one way to tile the first row



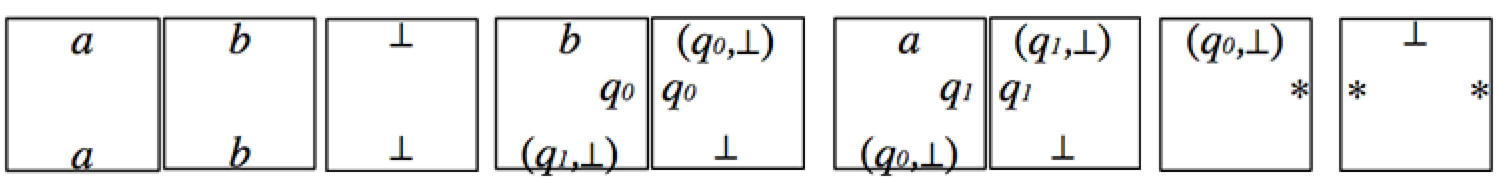
Initial configuration of the machine

Time:



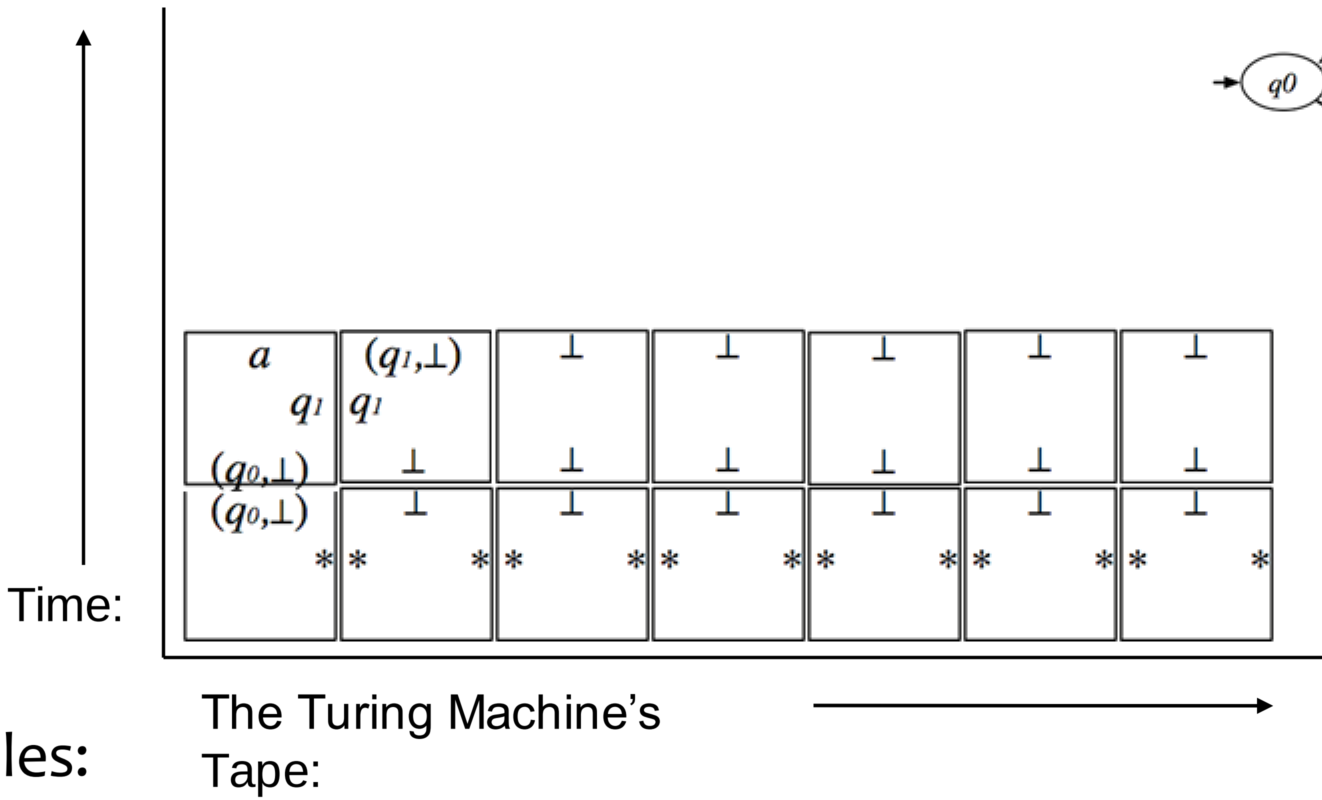
The Turing Machine's Tape:

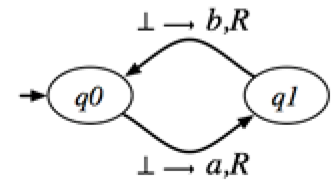
Set of Tiles:



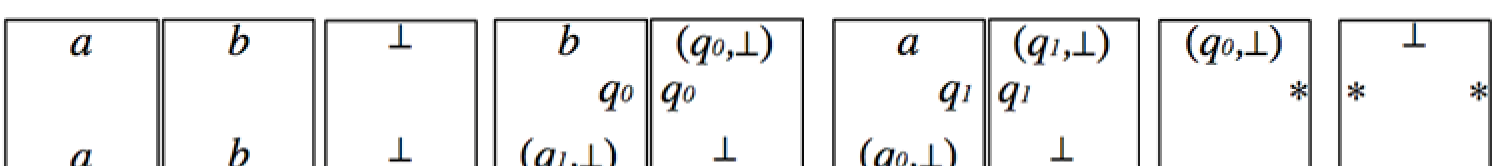
Claim: We can tile the plane ⇔ M(ε) loops forever

Only one tile with bottom color $(q_0, \perp)$
Only one tile with left color q_1, bottom color $\perp$





Set of Tiles:

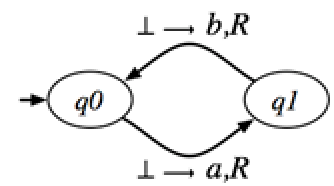


Claim: We can tile the plane ⇔ **M(ε)** loops forever

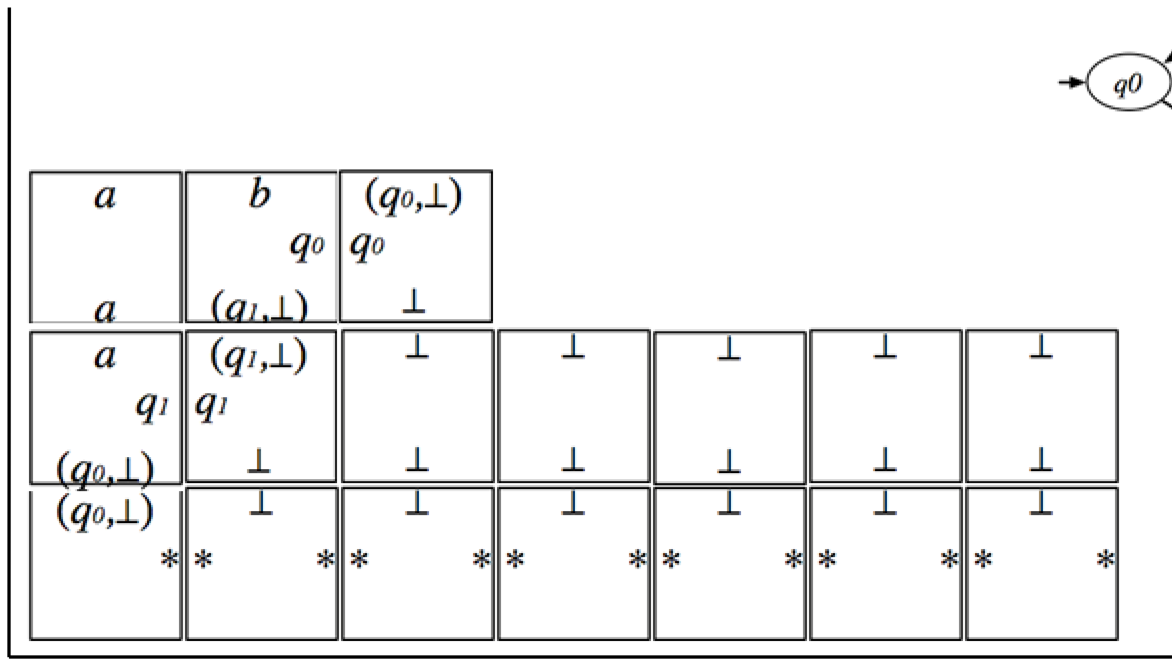
Only one tile with bottom color a

Only one tile with bottom color $(q_1, \perp)$

Only one tile with left color q_0, bottom color $\perp$

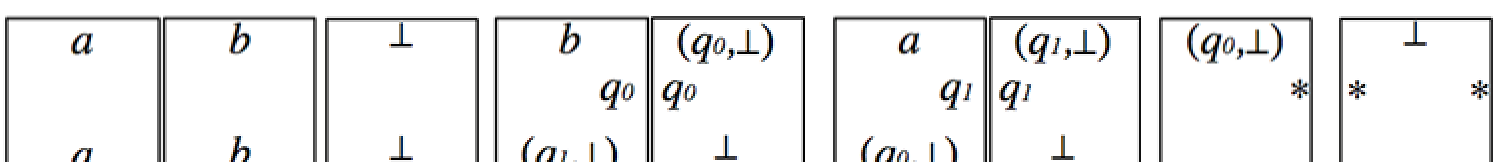


Time:

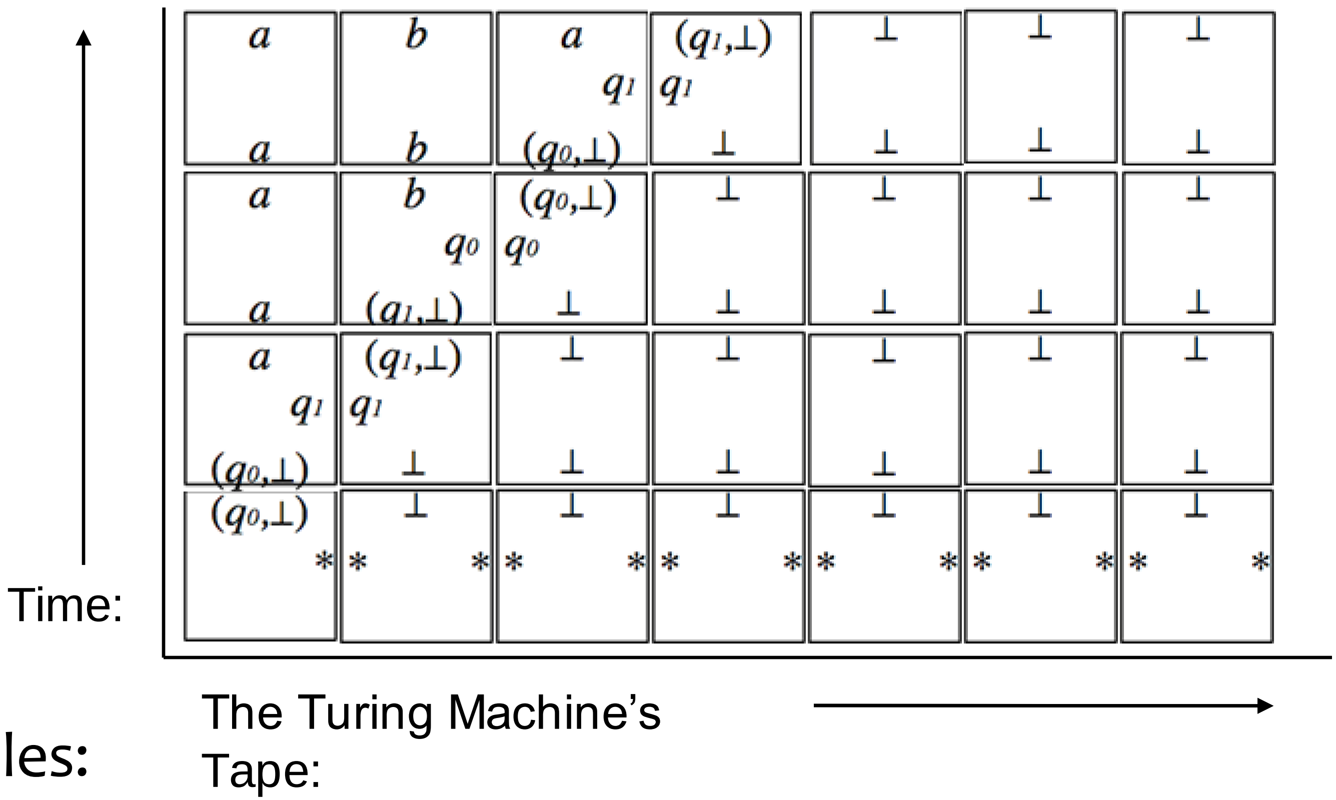


The Turing Machine's Tape:

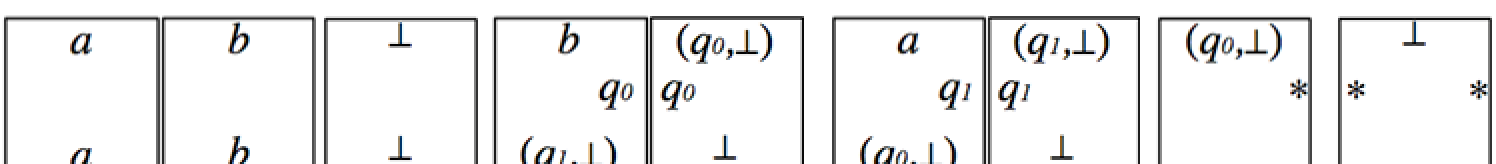
Set of Tiles:



Claim: We can tile the plane ⇔ **M(ε)** loops forever



Set of Tiles:



List of undecidable problems

2 languages

Article Talk Tools

From Wikipedia, the free encyclopedia

In computability theory, an **undecidable problem** is a type of computational problem that requires a yes/no answer, but where there cannot possibly be any computer program that always gives the correct answer; that is, any possible program would sometimes give the wrong answer or run forever without giving any answer. More formally, an undecidable problem is a problem whose language is not a recursive set; see the article Decidable language. There are uncountably many undecidable problems, so the list below is necessarily incomplete. Though undecidable languages are not recursive languages, they may be subsets of Turing recognizable languages: i.e., such undecidable languages may be recursively enumerable.

Many, if not most, undecidable problems in mathematics can be posed as word problems: determining when two distinct strings of symbols (encoding some mathematical concept or object) represent the same object or not.

For undecidability in axiomatic mathematics, see List of statements undecidable in ZFC.

Problems in logic [edit]

- Hilbert's Entscheidungsproblem.
- Type inference and type checking for the second-order lambda calculus (or equivalent).[1]
- Determining whether a first-order sentence in the logic of graphs can be realized by a finite undirected graph.[2]
- Trakhtenbrot's theorem - Finite satisfiability is undecidable.
- Satisfiability of first order Horn clauses.

Problems about abstract machines [edit]

- The halting problem (determining whether a Turing machine halts on a given input)

Problems about matrices [edit]

- The mortal matrix problem: determining, given a finite set of $n \times n$ matrice
integer entries, whether they can be multiplied in some order, possibly wi
to yield the zero matrix. This is known to be undecidable for a set of six o
matrices, or a set of two 15×15 matrices.[3]
- Determining whether a finite set of upper triangular 3×3 matrices with no
integer entries generates a free semigroup.
- Determining whether two finitely generated subsemigroups of integer ma
common element.

Problems in combinatorial group theory [edit]

- The word problem for groups.
- The conjugacy problem.
- The group isomorphism problem.

Problems in topology [edit]

Main article: Simplicial complex recognition problem

- Determining whether two finite simplicial complexes are homeomorphic.
- Determining whether a finite simplicial complex is (homeomorphic to) a m
- Determining whether the fundamental group of a finite simplicial complex
- Determining whether two non-simply connected 5-manifolds are homeom
a 5-manifold is homeomorphic to S^5.[4]

Problems in analysis [edit]

- For functions in certain classes, the problem of determining: whether two
are equal, known as the zero-equivalence problem (see Richardson's the
zeroes of a function; whether the indefinite integral of a function is also in