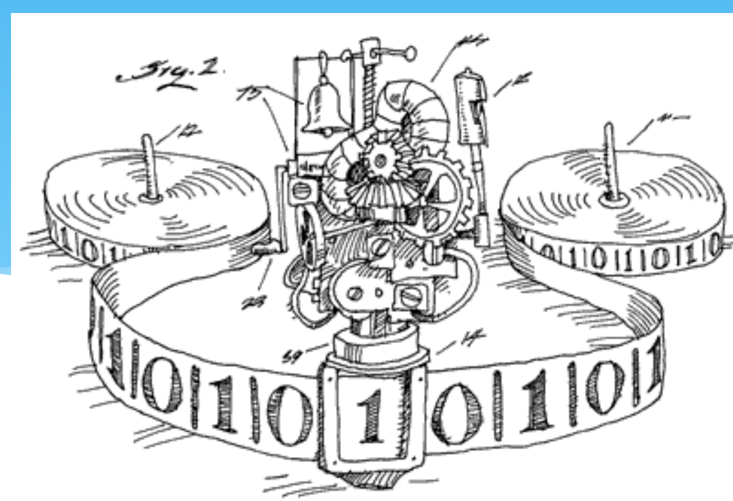
EECS 376: Foundations of Computer Science

Chris Peikert

8 February 2023



Today's Agenda

* Recap: Barber Paradox, Undecidability of $L_{\text{ACC}}, L_{\text{HALT}}$
* Turing reducibility and its consequences
* More undecidable problems for Turing Machines
* Undecidable problems that don't (seem to) relate to TMs

(Computational) Barber Paradox

* **Sign:** *"Barber B is the best barber in town! B cuts the hair of all those—and only those—who do not cut their own hair."*
* Let's consider a computational analogy, where:
  * barber, people $\Rightarrow$ program(s)
  * hair $\Rightarrow$ source code
  * cut $\Rightarrow$ accept
* **Result:** *"Program B accepts the source code of all programs—and only those programs—that do not accept their own source code."*
* **Reminder:** The ***language of a program*** is the set of inputs it *accepts*.

  Thus, $L(B) = \{\langle M \rangle : M \text{ does not accept } \langle M \rangle\}$

Computational Barber Paradox

* **Result:** *"Program B accepts the source code of all programs—and only those programs—that do not accept their own source code."*

* **Question:** Does program B accept its own source code?
* **Answer:** Suppose P is a program.

1. P accepts its own code $\Longrightarrow$ B does not accept P's code.
2. P does not accept its own code $\Longrightarrow$ B accepts P's code.

* **Question:** What if $P = B$?

1. B accepts its own code $\Longrightarrow$ B does not accept B's code.
2. B does not accept its own code $\Longrightarrow$ B accepts B's code.

Contradiction! **B doesn't exist**; no machine decides L_{BARBER}

L_{ACC}: A "Useful" Language

Could we have a program that, given a Turing Machine M and string x, determines whether M accepts x ?

I.e., is this language decidable?

$L_{\mathsf{ACC}} = \{\langle M, x\rangle : M \text{ accepts } x\}$

* Any decider C for L_{ACC} must behave as follows:
  * M accepts $x \Longrightarrow C$ accepts $(\langle M\rangle, x)$
  * M does not accept $x \Longrightarrow C$ rejects $(\langle M\rangle, x)$

We showed: if such a C exists, we can use it to decide L_B.

Contradiction!: L_B is undecidable. So L_{ACC} is undecidable too.

L_{ACC} is Undecidable

We need to implement:

B takes one input: $\langle M \rangle$

M does not accept $\langle M \rangle \implies B$ accepts $\langle M \rangle$

M accepts $\langle M \rangle \implies B$ rejects $\langle M \rangle$

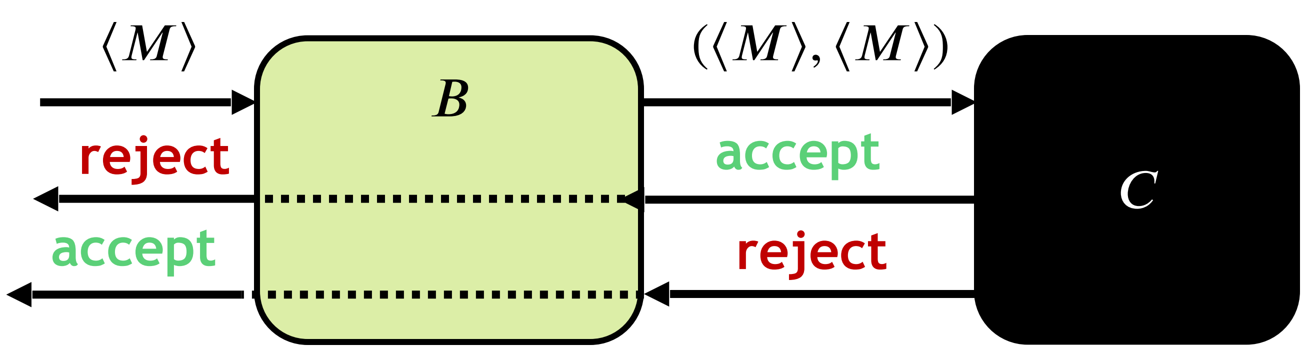
We have:

C takes two inputs: $\langle M \rangle$ and x.

M accepts $x \implies C$ accepts $(\langle M \rangle, x)$

M doesn't accept $x \implies C$ rejects $(\langle M \rangle, x)$

- **Proof:** Assume (for contradiction) that a decider C exists for $L_{\text{ACC}} = \{(\langle M \rangle, x) : M \text{ accepts } x\}$.
  We will use C to construct a decider B for $L_{\text{BARBER}} = \{\langle M \rangle : M \text{ does not accept } \langle M \rangle\}$:



Halting Problem

Halting Problem: Given a TM M and string x as input, decide if M halts on x.

We showed: if L_{HALT} is decidable, then we can build a decider for L_{ACC}. Contradiction!

(As we shall see, L_{ACC} and L_{HALT} are very useful for showing undecidability of even more problems.)

To prove that language L is undecidable: use a hypothetical decider for L to design a decider for L_{ACC} (say).

Requires some insight, but a few tricks go a long way…

L_{HALT} is Undecidable

We need to implement:

C is given two inputs: $\langle M \rangle$ and x

M accepts $x \implies C$ accepts $(\langle M \rangle, x)$

M does not accept x $\implies C$ rejects $(\langle M \rangle, x)$

We have:

H takes two inputs: $\langle M \rangle$ and x

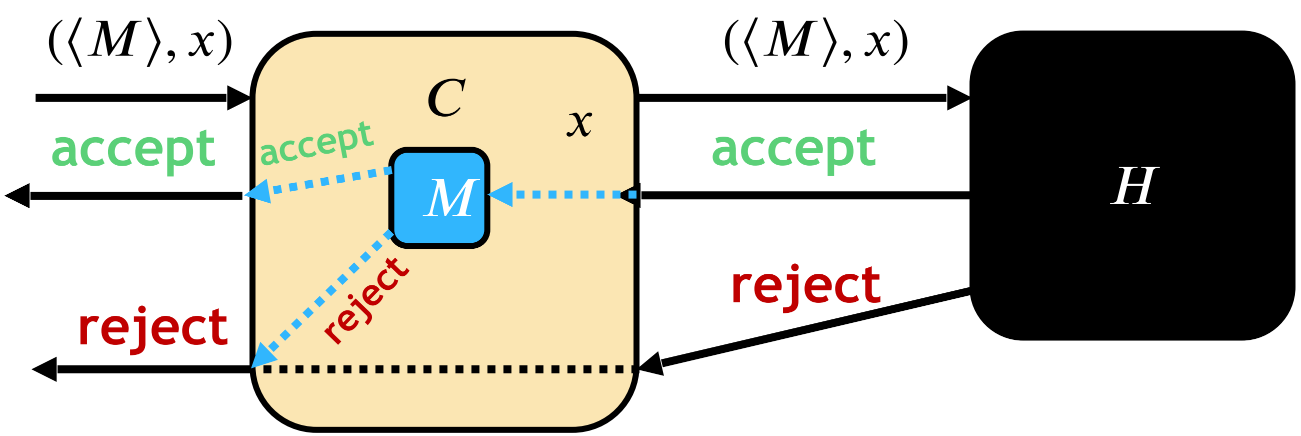
M accepts/rejects $x \implies H$ accepts $(\langle M \rangle, x)$

M loops on $x \implies H$ rejects $(\langle M \rangle, x)$

* **Claim:** $L_{\text{HALT}} = \{(\langle M \rangle, x) : M \text{ halts on } x\}$ is undecidable.
* **Proof:** Assume (for contradiction) that some H decides L_{HALT}.

  We construct a decider C for $L_{\text{ACC}} = \{(\langle M \rangle, x) : M \text{ accepts } x\}$:

Trick: "safely" run M(x) inside C



Reducibility

- **Definition:** Language A is ***Turing reducible*** to language B, written $\boldsymbol{A \le_T B}$, if there exists a TM that decides A given a *membership oracle* for B (a "black box").
  - The oracle correctly answers any query "is $x \in B$?"

- **Intuition:** B is "*no easier*" to solve than A is.
- **Previous results rephrased:** $L_{\text{BARBER}} \le_T L_{\text{ACC}} \le_T L_{\text{HALT}}$.

(The order can be confusing at first, and it is easy to make mistakes.
To prove L_{HALT} undecidable, we show: decider for $L_{\text{HALT}} \Rightarrow$ decider for L_{ACC}.
I.e., we "**reduce**" the task of deciding L_{ACC} to that of deciding L_{HALT}.)

Consequences of Reducibility

* **Theorem:** If $A \leq_T B$ and B is decidable, then A is decidable.
* **Proof:** implement the oracle ("black box") using a decider for B. This yields an ordinary TM that decides A.

* **Corollary:** If $A \leq_T B$ and A is undecidable, then B is undecidable.

* **Strategy:** Pick an undecidable language A and show that $A \leq_T B$.

More Undecidable Problems

- **Q:** Is $L_{\varepsilon\text{-HALT}} = \{\langle M \rangle : M \text{ halts on input } \varepsilon\}$ decidable?

- It looks "easier" than L_{HALT}, since it asks about just a *single input* ε rather than an *arbitrary* input x.
  Indeed, $L_{\varepsilon\text{-HALT}} \leq_T L_{\text{HALT}}$. (Why?)

- **Caution:** That reduction doesn't answer the above question, because it goes in the 'wrong direction'!
- To prove that $L_{\varepsilon\text{-HALT}}$ is undecidable, we can instead show that $L_{\text{HALT}} \leq_T L_{\varepsilon\text{-HALT}}$.

$L_{\text{HALT}} \leq_T L_{\varepsilon-\text{HALT}}$

We need to implement:

H takes two inputs: $\langle M \rangle$ and x

M halts on $x \implies H$ accepts $\langle M \rangle, x$

M loops on $x \implies H$ rejects $\langle M \rangle, x$

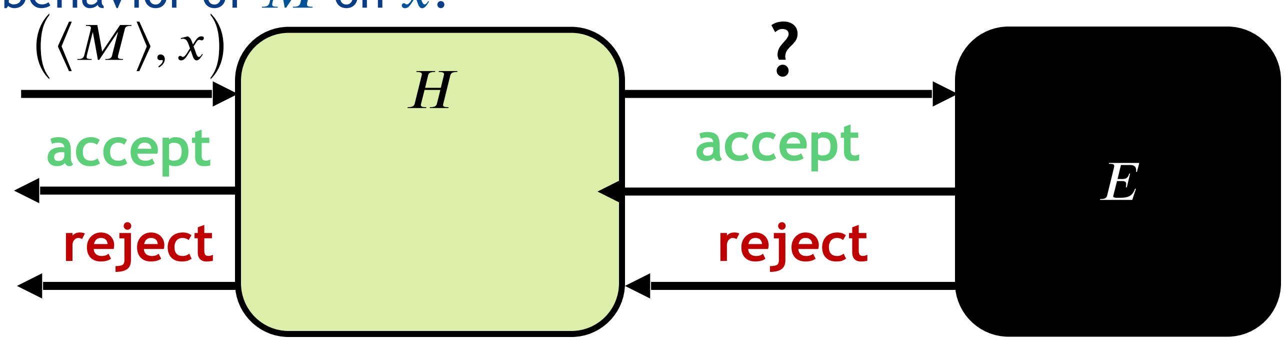
We have:

E takes **one** input: $\langle M \rangle$

M halts on $\varepsilon \implies E$ accepts $\langle M \rangle$

M loops on $\varepsilon \implies E$ rejects $\langle M \rangle$

* **Task:** Let E be a membership oracle for $L_{\varepsilon-\text{HALT}}$.
  We need to construct a decider H for L_{HALT}.
* E only tells us whether a queried program halts on ε. What program should we ask it about, to determine whether M halts on the given x?
* **New Idea:** Pass a *different program* to E than M itself!
  Have H construct a program M' whose behavior on ε matches the behavior of M on x.



$L_{\text{HALT}} \leq_T L_{\varepsilon\text{–HALT}}$

We need to implement:

H takes two inputs: $\langle M \rangle$ and x

M halts on $x \implies H$ accepts $\langle M \rangle, x$

M loops on $x \implies H$ rejects $\langle M \rangle, x$

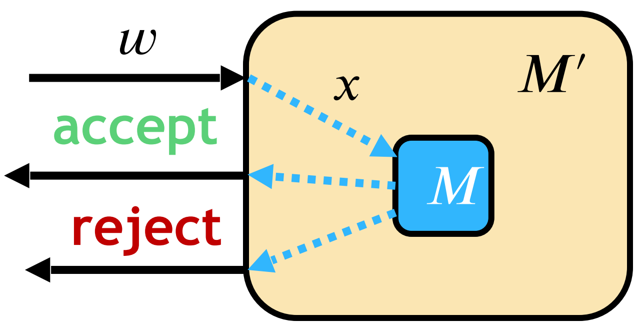
We have:

E takes **one** input: $\langle M \rangle$

M halts on $\varepsilon \implies E$ accepts $\langle M \rangle$

M loops on $\varepsilon \implies E$ rejects $\langle M \rangle$

- Have H construct a program M' whose behavior on ε matches the behavior of M on x.
- Program M': ignore input, run M on x and answer as M does.




```
Key Point: H constructs M' "live".
So H can hard-code its given M,x
into M'.
```

Q: What is the language of M' ?

$L_{\mathrm{HALT}} \leq_T L_{\varepsilon-\mathrm{HALT}}$

We need to implement:

H takes two inputs: $\langle M \rangle$ and x

M halts on $x \implies H$ accepts $\langle M \rangle, x$

M loops on $x \implies H$ rejects $\langle M \rangle, x$

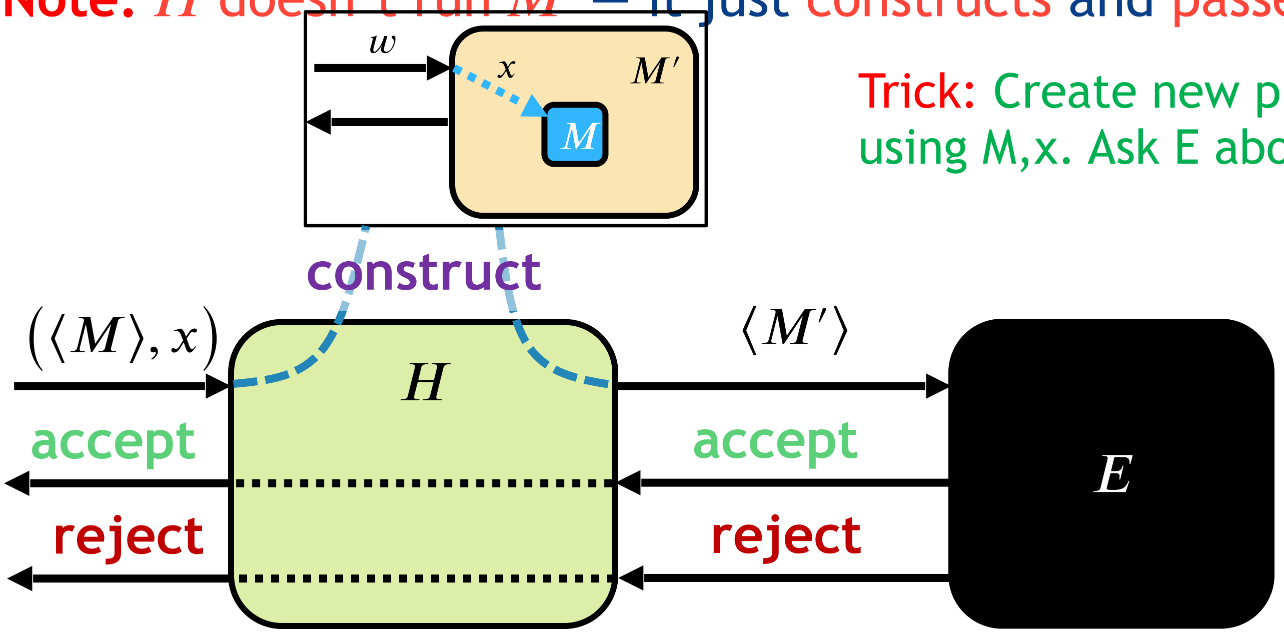
We have:

E takes **one** input: $\langle M \rangle$

M halts on $\varepsilon \implies E$ accepts $\langle M \rangle$

M loops on $\varepsilon \implies E$ rejects $\langle M \rangle$

* Have H construct a program M' whose behavior on ε matches the behavior of M on x.
* **Note:** H doesn't run M' – it just constructs and passes it to E.



Trick: Create new program M' using M,x. Ask E about M'.

$L_{\text{HALT}} \leq_T L_{\varepsilon-\text{HALT}}$

We need to implement:

H takes two inputs: $\langle M \rangle$ and x

M halts on $x \Longrightarrow H$ accepts $\langle M \rangle, x$

M loops on $x \Longrightarrow H$ rejects $\langle M \rangle, x$

We have:

E takes **one** input: $\langle M \rangle$

M halts on $\varepsilon \Longrightarrow E$ accepts $\langle M \rangle$

M loops on $\varepsilon \Longrightarrow E$ rejects $\langle M \rangle$

* **Proof:** Let E be a membership oracle for $L_{\varepsilon-\text{HALT}}$.
  We construct a decider H for L_{HALT}:
* $H\big(\langle M \rangle, x\big)$:
  1. Construct a program "$M'(w)$: run M on x and answer as M does" *(hard-code M,x)*
  2. Run E on $\langle M' \rangle$ and answer as E does *(accept if E accepts, else reject)*

* **Analysis:** H halts on any input (it can construct M', and E halts on $\langle M' \rangle$).
  Moreover:
  * M halts on $x \iff M'$ halts on $\varepsilon \iff E$ accepts $\langle M' \rangle \iff H$ accepts $\big(\langle M \rangle, x\big)$
* **Conclusion:** H is a decider for L_{HALT}, so $L_{\text{HALT}} \leq_T L_{\varepsilon-\text{HALT}}$.
  Since L_{HALT} is undecidable, so is $L_{\varepsilon-\text{HALT}}$

Post Mortem

- H constructs a program M' that either:
  1. halts on *every* input (including ε), or
  2. loops on *every* input (including ε),

  depending on whether M halts on x.

Then it asks the $L_{\varepsilon\text{-HALT}}$ membership oracle which of these is the case.

Observe: There is nothing special about the empty string ε here!

Exercise: Show that this language is undecidable:

$L_{376-HALT} = \{\langle M \rangle : M \text{ halts on input string "376"}\}$

Autograder (Equal Langs Problem)

We need to implement:

C takes two inputs: $\langle M \rangle$ and x

M accepts $x \Longrightarrow C$ accepts $\langle M \rangle, x$

M does not accept $x \Longrightarrow C$ rejects $\langle M \rangle, x$

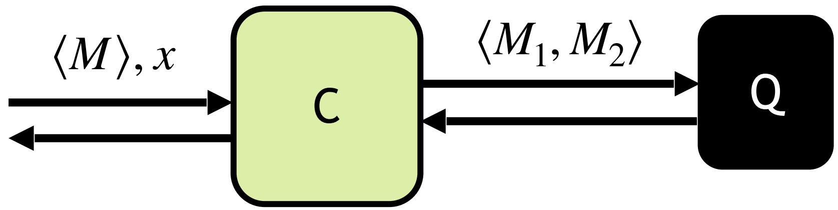
We have:

Q takes two inputs: $\langle M_1, M_2 \rangle$

$L(M_1) = L(M_2) \Longrightarrow Q$ accepts $\langle M_1, M_2 \rangle$

$L(M_1) \neq L(M_2) \Longrightarrow Q$ rejects $\langle M_1, M_2 \rangle$

- **Claim:** $L_{\mathrm{EQ}} = \{\langle M_1, M_2 \rangle : L(M_1) = L(M_2)\}$ is undecidable.
- **Proof:** We show $L_{\mathrm{ACC}} \leq_T L_{\mathrm{EQ}}$. Let Q be a membership oracle for L_{EQ}. We construct a decider C for L_{ACC}.



- **Idea:** C fixes M_2 to accept every string, so $L(M_2) = \Sigma^*$.
- Then C constructs M_1 where:
  - M accepts $x \Rightarrow L(M_1) = \Sigma^*$
  - M does not accept $x \Rightarrow L(M_1) \neq \Sigma^*$
- "$M_1(w)$: ignore input, run $M(x)$ and answer as M does."

Conclusion:
There is no universal autograder!

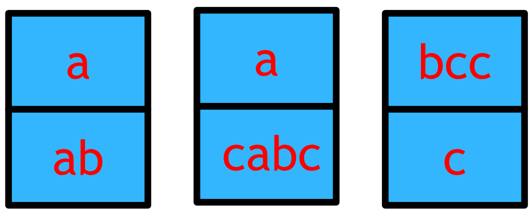
Do All Undecidable Problems Involve Turing Machines?

No!

Many other “natural” problems are undecidable, despite not seeming to be “about” computation.

Post’s Correspondence Problem

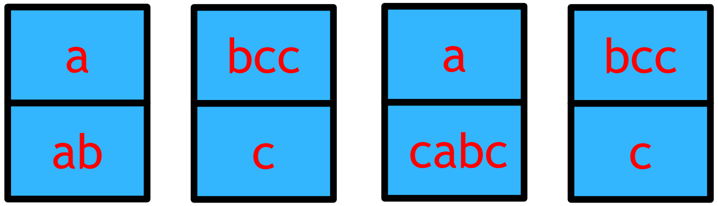
* Input: *finite* set of “dominoes” with strings written on each half.
* E.g.:



Problem: Is there a sequence of these dominoes (repetitions allowed) for which top string = bottom string?

* Match: =

abccabcc = abccabcc



Post's Correspondence Problem

Theorem (Post 1946): This problem is undecidable:
There is no general algorithm to determine if a match is possible!

Key proof ideas:

- Reduction $L_{\text{ACC}} \leq_T L_{\text{PCP}}$: given $\langle M \rangle$ and x, design "dominoes" so that a "match" is equivalent to the "history" of an accepting run of $M(x)$.
- Dominoes correspond to initial tape contents, effects of M's transition rules, and ending in the accept state.
- They only "fit together" in a way that mirrors the execution of $M(x)$.
- Membership oracle for L_{PCP} reveals whether $M(x)$ accepts!

Mortal Matrix Problem

Given two 15-by-15 matrices A and B:

Is it possible to multiply A and B together (in any order, with repetitions allowed) to get an all-0s matrix?

Theorem (Cassaigne, Halava, Harju, Nicolas 2014):

The problem is undecidable!

Richardson Problem

Input: A set S of rational numbers.

You can build an expression E from the numbers in S, the numbers π and $\ln(2)$, the variable x, and operations $+$, $-$, $\cdot$, $\circ$, sin, exp, abs.

Question: Can you make an E such that $E(x) = 0$ for all x?

Theorem (Richardson, 1968): This problem is undecidable!

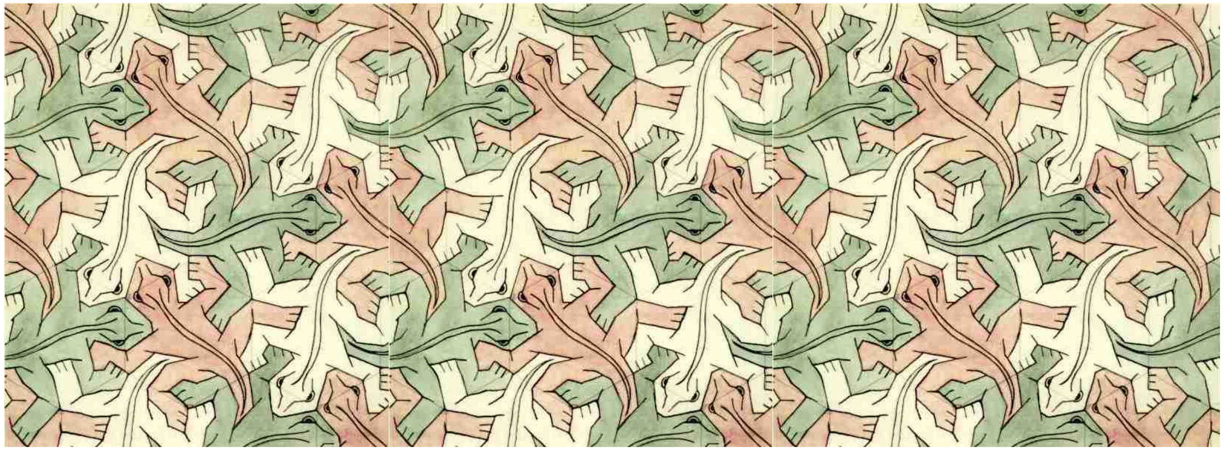
Many More…

Hilbert's 10th problem: given a polynomial in finitely many variables with integer coefficients, does it have an integral solution? Undecidable!

Wikipedia list of many more: https://en.wikipedia.org/wiki/List_of_undecidable_problems

Tiling the Plane

- S is a finite set of shapes.
- There is an infinite supply of each shape.
- **Question:** Can we "tile the plane" using these shapes? (No overlaps or gaps allowed.)

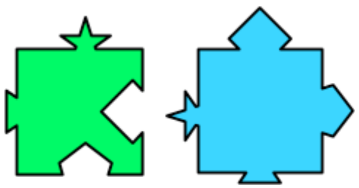


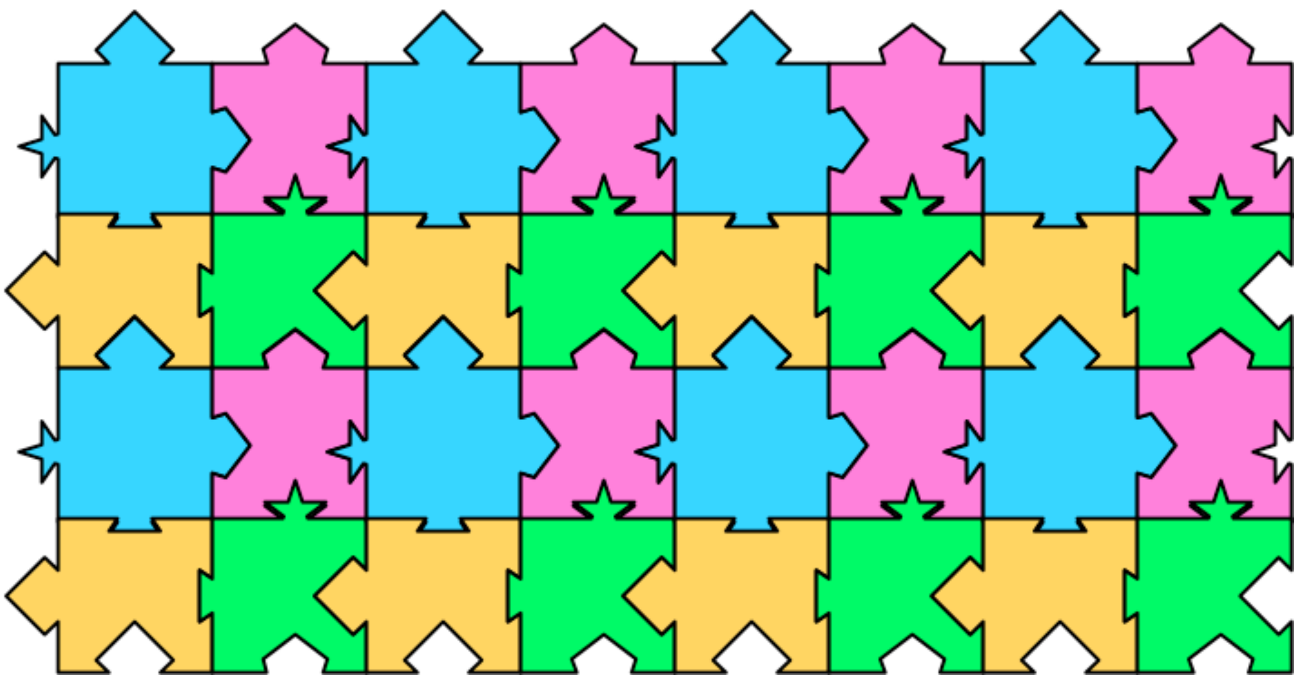
Tiling the Plane

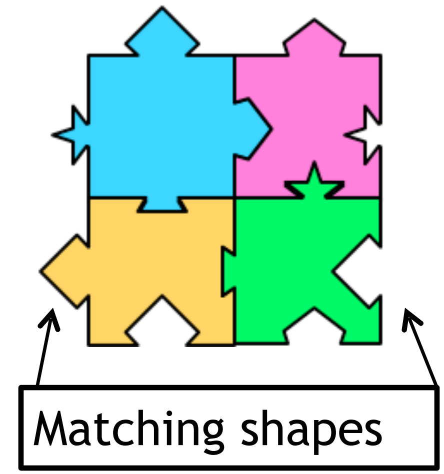
- S is a finite set of shapes.
- There is an infinite supply of each shape.
- **Question:** Can we "tile the plane" using these shapes? (No overlaps or gaps allowed.)

- **Formally:** $L_{\text{TILE}} = \{\langle T \rangle : T$ **is a set of shapes that tiles the plane**$\}$

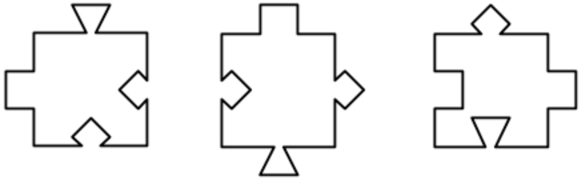
Example

* T = 



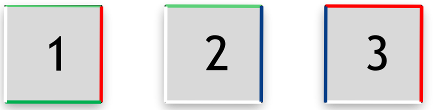


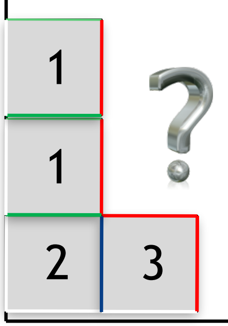
Example

- T = 
- **Question:** Is it possible for a computer to determine if a given set T tiles the plane?
- **Formally:** Is L_{TILE} decidable?
- **Answer:** No!
- **Idea:** Tiling can simulate a TM: we can use an L_{TILE} oracle to solve the Halting Problem.

Wang Tilings [Hao Wang 1966]

1. Consider the positive quadrant of a plane
2. All tiles are square, each side has a "color"
3. Two squares can be adjacent only if their colors match
4. Squares cannot be rotated or flipped
5. The boundary of the quadrant is colored white





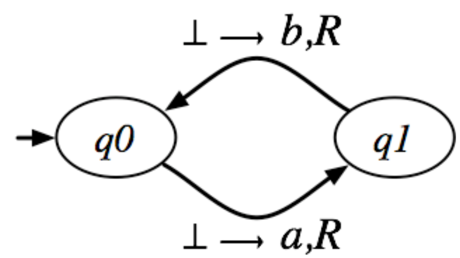
Surprisingly, undecidable even when no choice is involved;
(exactly one valid choice for tile at each step)

Tiling the Plane is Undecidable

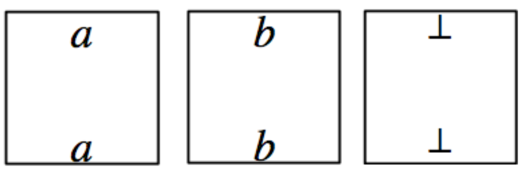
* **General Idea:** Given a TM M, construct a set of tiles T s.t. T can tile the plane $\iff$ M does not halt on empty string:
  * Each row can be tiled in exactly one way, given the previous row.
  * The i^{th} row encodes the tape of the TM after i steps.
  * If the TM does not halt, the quadrant has a unique tiling.
  * If the TM does halt, the quadrant cannot be tiled.
* **Formally:** $L_{\varepsilon\text{-HALT}} \leq_T L_{\text{TILE}}$
* **Conclusion:** L_{TILE} is undecidable

Converting a TM to a Set of Tiles

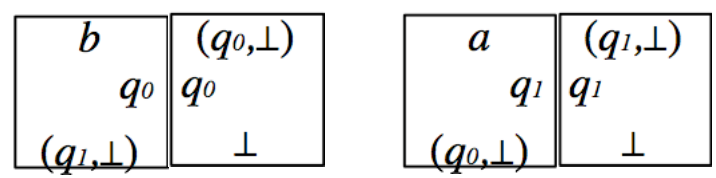
- Consider the following TM:
  (with no accept or reject states)



- Make one tile for each symbol in Γ:

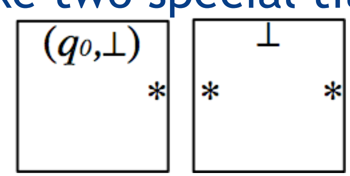


- Make two tiles for each transition:

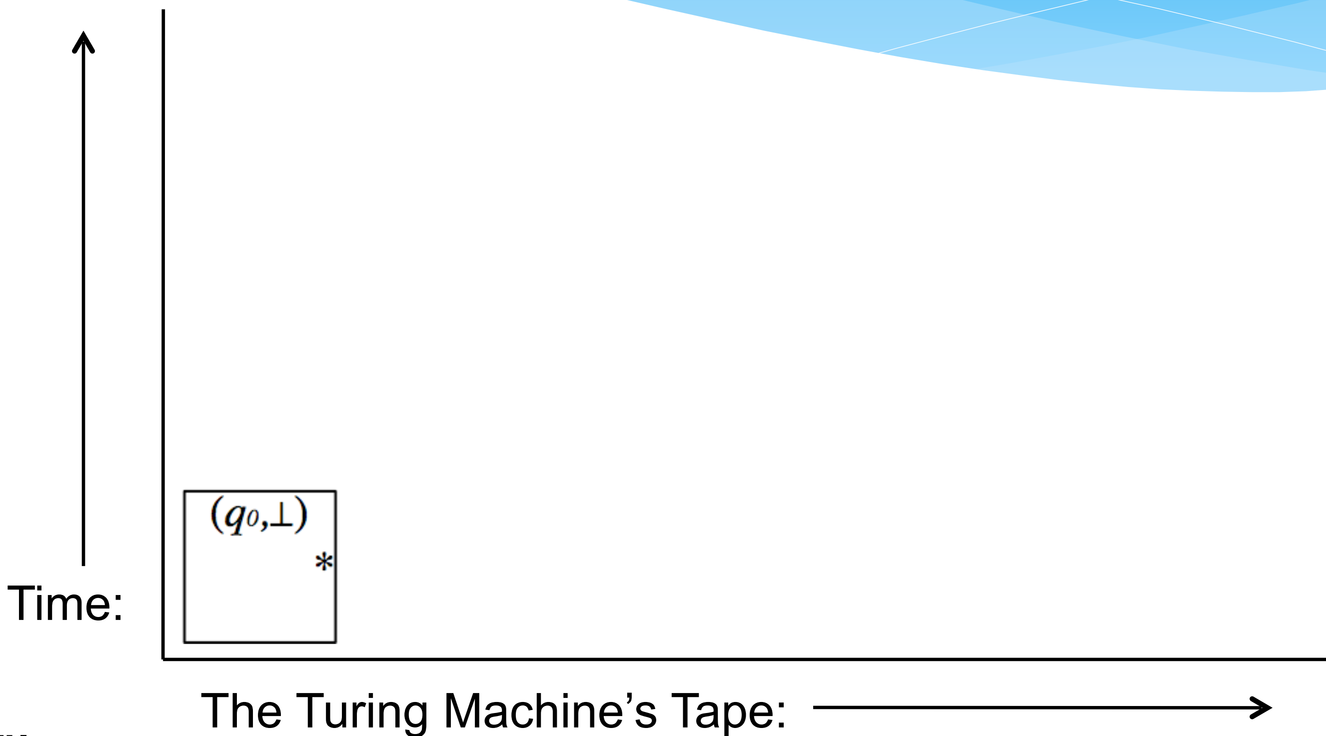


Transitions for the TM above

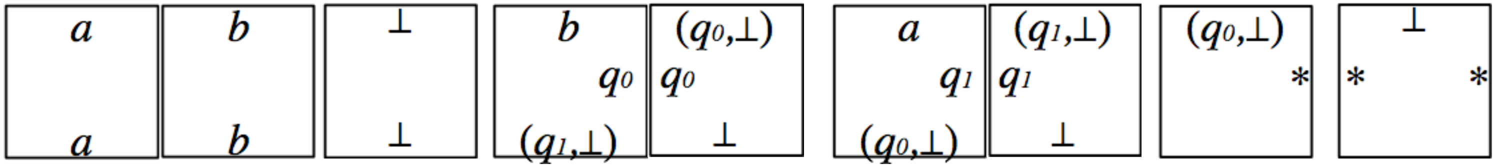
- Make two special tiles for the start state and $\perp$ symbol:



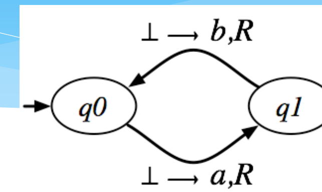
Only one tile is white on both corner edges



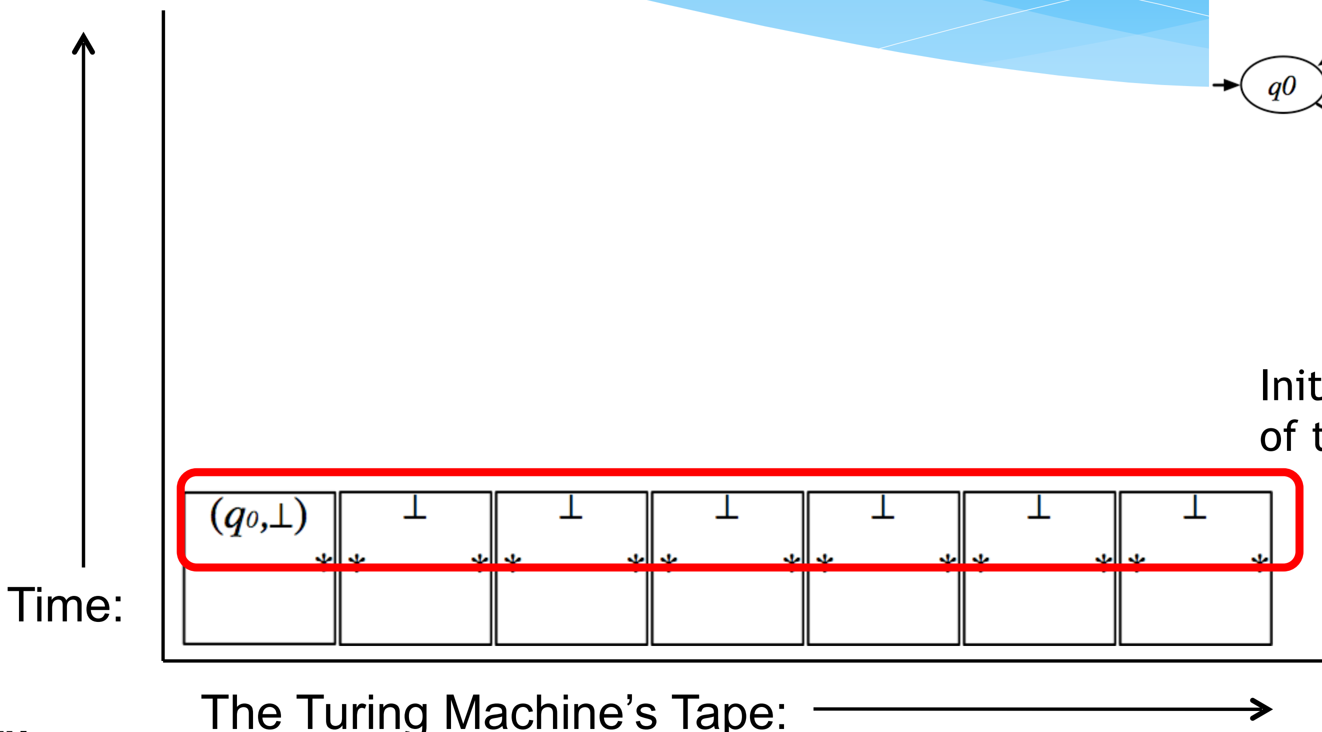
Set of Tiles:



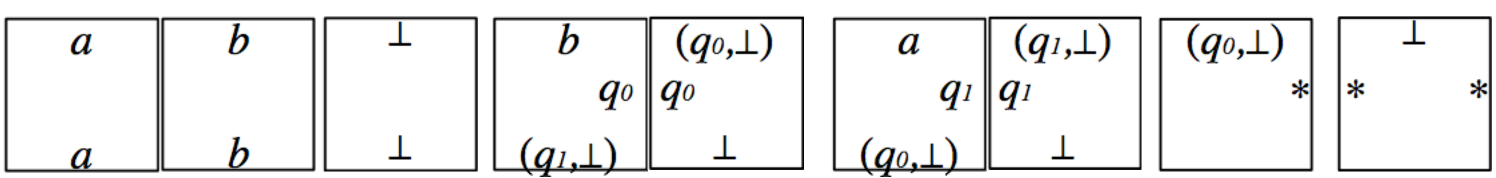
Only one way to tile the first row



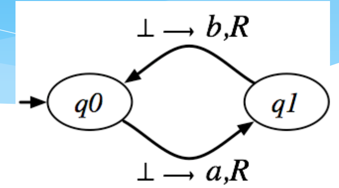
Initial configuration of the machine

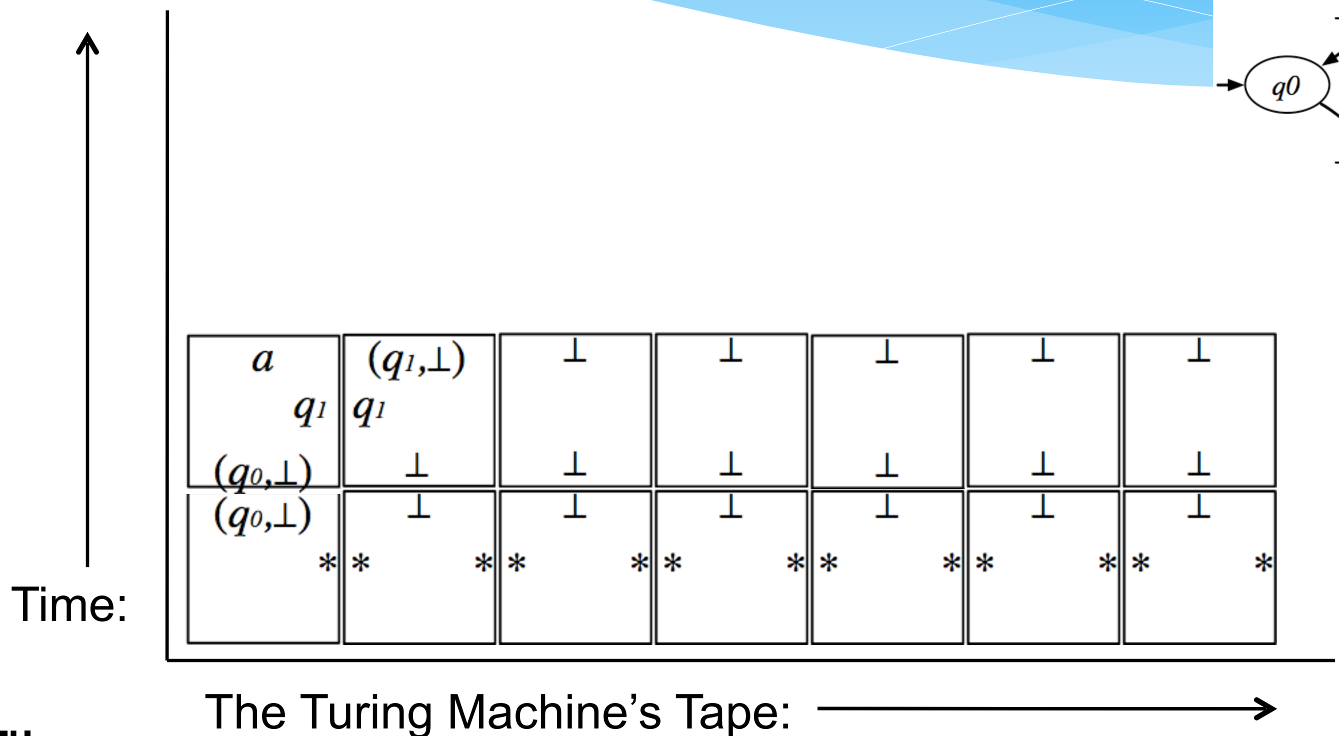


Set of Tiles:

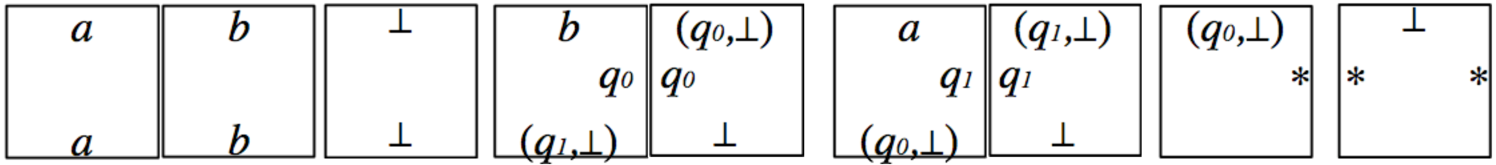


Only one tile with bottom color $(q_0, \perp)$
Only one tile with right color q_1, bottom color $\perp$



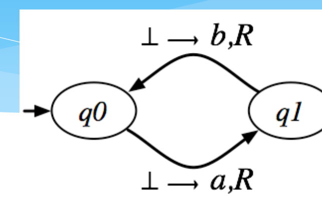


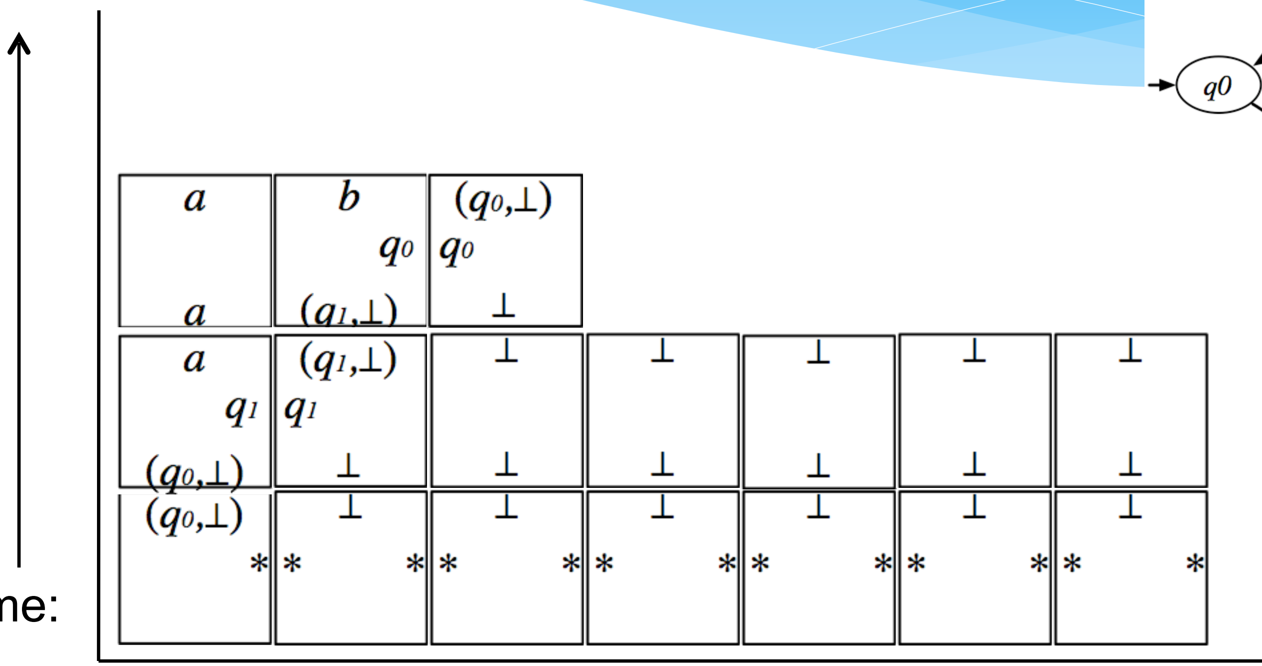
Set of Tiles:



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

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

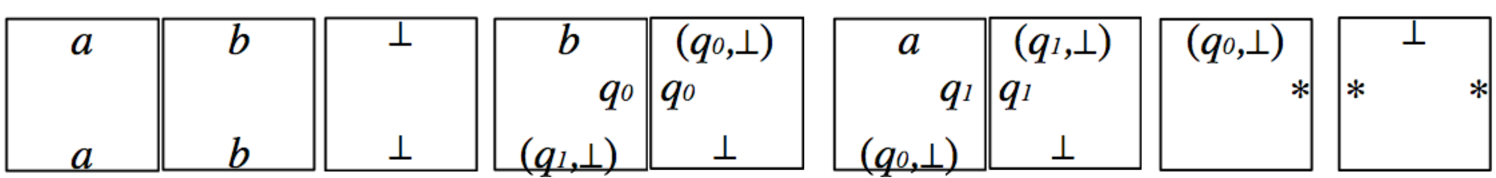




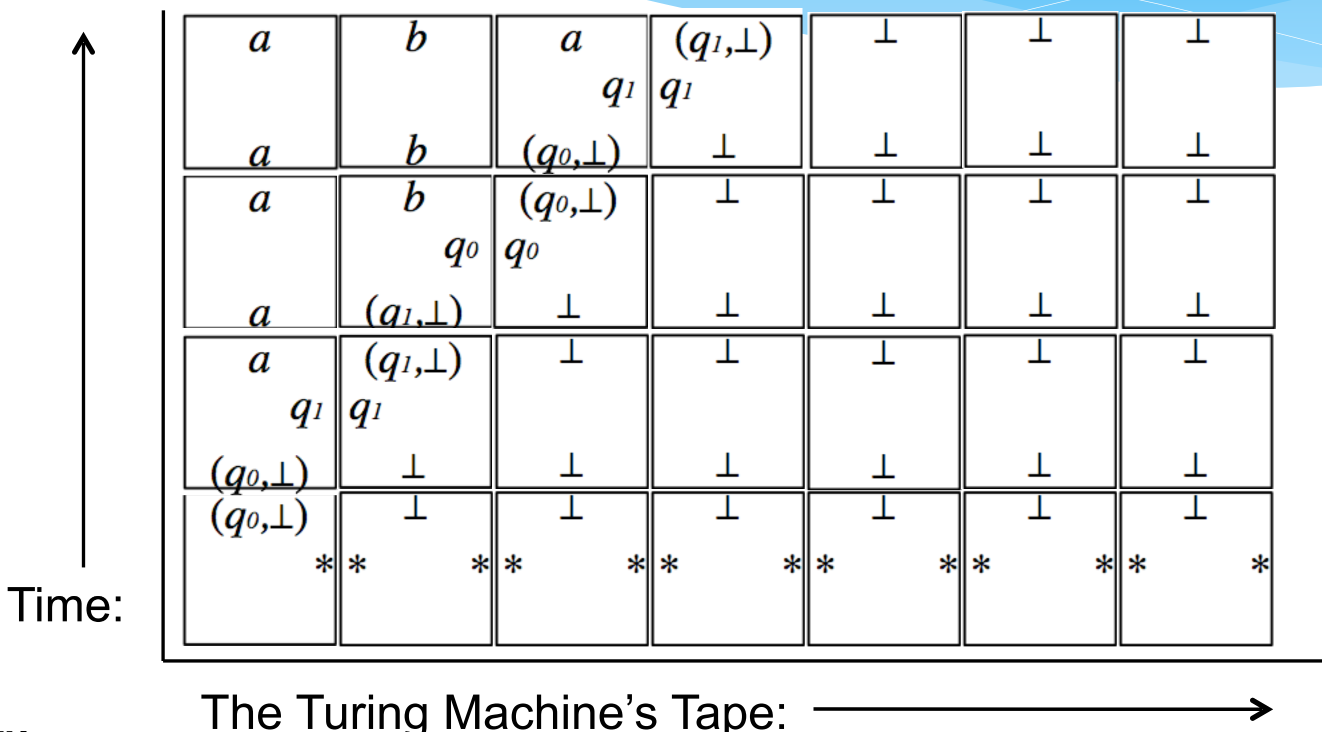
Time:

The Turing Machine's Tape:

Set of Tiles:



Etc ...



Set of Tiles: