then:

on input $b(M) * w$ ($*$ - blank)
U returns what M does.

↳ we call U a universal Turing machine:

The Halting Problem

- we can use the universal TM U to show that a certain decision problem is undecidable

- the construction may look like a bit of a maguetrick, but is an instance of what's called a diagonalization argument
  ↳ such arguments are ubiquitous in computability theory and very powerful.

- In what follows (as above) we'll believe that Turing machines are powerful enough to carry out algorithms we can write down informally (or know to exist already)

Def'n The **Halting Problem** is the decision problem (A, P) where the alphabet is $A = \{0,1\}$
and $P = \{b(M) : M$ is a TM on $\{0,1\}$ that terminates on the empty input$\}$

↳ i.e. the problem is to determine, given a code $b(M)$ for a TM M, whether M halts on $\emptyset$.

Theorem (Turing) The halting problem is undecidable.

PF - Let U denote the universal TM
- for each binary string $x \in \{0,1\}^*$ let U_x denote the TM that, on the empty input $\emptyset$:
  ① prints $x*x$ on its tape
  ② moves the head over the first symbol (in the first x)
  ③ runs U on this input (i.e. on $x*x$)

- it is "not hard" ← can be done mechanically to program U_x from the table for U (for a fixed x)
- in particular we can write a subroutine for a Turing machine that, on input x, outputs the table for U_x ... in the form of $b(U_x)$

- Notation: For a TM M, we'll write:
  $M(w)\downarrow$ if M halts on input w
  $M(w)\uparrow$ if M does not halt on input w

- Now: spse toward a contradiction the halting problem were decidable

- then in particular we could, for a given x, decide (algorithmically) whether U_x halts on $\emptyset$

- so we could write a TM M_0 that, on input x,
  - decides whether U_x halts on $\emptyset$ (using our algorithm)
  - terminates w/ output 1 if U_x does not terminate on $\emptyset$
  - does not terminate (by e.g. going into some back-and-forth loop) if U_x does terminate on $\emptyset$

↑ we're "diagonalizing"

- i.e. we have

$M_0(x)\downarrow$ iff $U_x(\phi)\uparrow$

iff $U(x*x)\uparrow$

- so, for any TM M on $\{0,1\}$ we have

$M_0(b(M))\downarrow$ iff $U(b(M)*b(M))\uparrow$

iff $M(b(M))\uparrow$

- in particular for $M = M_0$ we get

$M_0(b(M_0))\downarrow$ iff $M(b(M_0))\uparrow$

a contradiction ✓

—

Roughly: here's a summary of the proof:

if we could decide (by a TM!) whether a given M halts on the empty input

then using U and some coding we could decide whether a given M halts on $b(M)$ (i.e. whether it halts when we input its own code)

then we could write an M_0 that halts on an input $b(M)$ iff M does not halt on $b(M)$

... so we get a "self-referential" or "diagonal" contradiction by plugging in $b(M_0)$ to M_0

"M_0 halts on $b(M_0)$ iff M_0 does not halt on $b(M_0)$" — impossible!

Other undecidable problems

1. VALIDITY

Recall. Sps L is a finite-first order language

$VALIDITY_L$ is the decision problem (A, P) where

$A = L \cup \{$other symbols needed to encode L-fmlas$\}$

P = set of valid L-fmlas

(i.e. $\varphi \in P$ iff $\vdash \varphi$)

Theorem (Church, Turing) There is a finite language L s.t. $VALIDITY_L$ is undecidable

2. Hilbert's 10th

Recall. a **Diophantine Equation** is an equation of the form $p(x_1, \ldots, x_n) = 0$ where p is a polynomial in the variables $x_1, \ldots, x_n$ w/ integer coeffs

- e.g. $x^2 + y^2 - z^2 = 0$ and $20xyz - w^2 + 2xy^2 = 0$ are Diophantine

Hilbert's 10th problem: is there an algorithm that takes as input a diophantine polynomial p and decides whether $p = 0$ has integer solutions?

- we can formulate this problem as a decision problem
- Consider the alphabet
  $A = \{x, 0, /, ', +, -, \wedge, (,)\}$

- if we encode a variable x_n by x(n written in binary),
  (... eg. x_3 by $x(11)$),
  write each natural number n in binary, and use $\wedge$ for exponentiation, then any diophantine polynomial p can be encoded as a word in A

- e.g. $x_1^2 - 7x_2^2 - 1$ is written
  $(x(1))\wedge(10) + (-111)(x(10)\wedge 10) + (-1)$

- now let DIOPHANTINE denote the decision problem (A, P) where
  $P = \{w \in A^* :$ w encodes a Diophantine polynomial which has an integer sol'n$\}$

- we have the following Fields medal winning theorem
  Theorem (Matiyasevich) DIOPHANTINE is undecidable

↪ this gives a negative soln to Hilbert's 10th!

Decidable Problems

CTCH: are there (interesting?) problems that can be decided algorithmically? (i.e by a TM / RM / computer / etc)

- we present some easy and not-so-easy examples
- will describe problems informally, i.e. w/o specifying details of alphabet.

Ex's ① SATISFIABILITY is the following problem:
- given a prop'l fmla φ, decide if φ is satisfiable.

↳ is clearly decidable
↳ algorithm: if $p_1, \ldots, p_n$ are the variables in φ, go thru all 2^n truth assignments and see if any satisfy φ

② TRAVELING SALESMAN is the following problem:
- given a finite set of "cities" $\{c_1, \ldots, c_n\}$, distances $d(c_i, c_j) \in \{1, 2, \ldots\}$ for all $i \neq j$, and a bound $B \in \{1, 2, \ldots\}$, decide if there is a tour of the cities of total distance $\leq B$.

... i.e. if there is an ordering $c_{i_1}, \ldots, c_{i_m}$ of $c_1, \ldots, c_m$ s.t. $d(c_{i_1}, c_{i_2}) + \cdots + d(c_{i_{m-1}}, c_{i_m}) \leq B$

↳ problem is also clearly decidable
↳ algorithm: list all possible orderings, calculate the length of the tour for each, check if any $\leq B$

- in general: problems that require checking only finitely many things (which only take finite time to check!) will be decidable.
- algorithm: check them all.
- what about problems that require possibly searching over an infinite space?

(8) ELEMENTARY ALGEBRA is the problem.

given a sentence σ in the language $L = \{0, 1, +, \cdot\}$, decide whether σ is true in the structure $\langle \mathbb{R}, 0, 1, +, \cdot \rangle$

Theorem (Tarski) ELEMENTARY ALGEBRA is decidable.

- a counterpoint to this result is the following

(Church)

Theorem The following problem:

"Determine whether a given sentence σ in the language $L=\{0, S, +, \cdot, <\}$ is true in $\langle \mathbb{N}, 0, S, +, \cdot, < \rangle$"

is undecidable!

Computational Complexity: the class P

- as ex's above show: there are lots of problems that are decidable, i.e. for which there are algorithms that *in principle* solve every instance of the problem.
- in practice we need algorithms that can solve problems *quickly*
- eg. if we have a computer running a program that solves our problem would like it to halt w/in ... a day? a week? our lifetime at least!

↳ ready to concept of **computational complexity**