David Hilbert

Hilbert's problems were presented in 1900 and were all unsolved at the time. These problems shaped a good portion of 20th century mathematics. These problems varied greatly in scope and precision. Some turned out to be solvable, some are still unsolved. Two created thriving mathematical subdisciplies.

10th problem

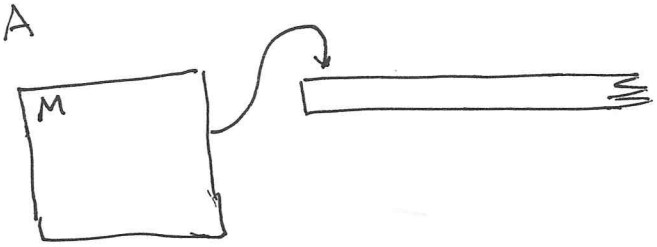
Find an algorithm to determine whether a given polynomial with 2 or more unknowns with integer coefficients has an integer solution.

It's now provable that no such algorithm exists, but why were they unable to prove that at the time?

They lacked a rigorous definition of an Algorithm.

Conceptual Models

DFA



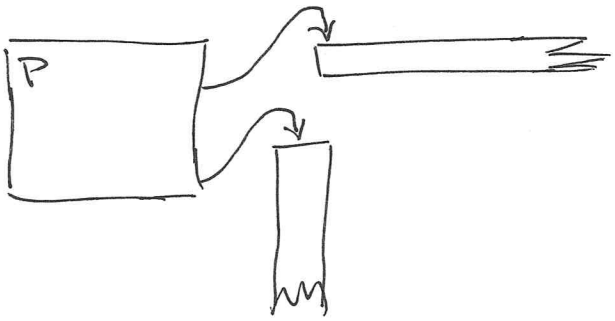
- Tape head moves only to the right
- Must move at every step

NFA

How is an NFA different?

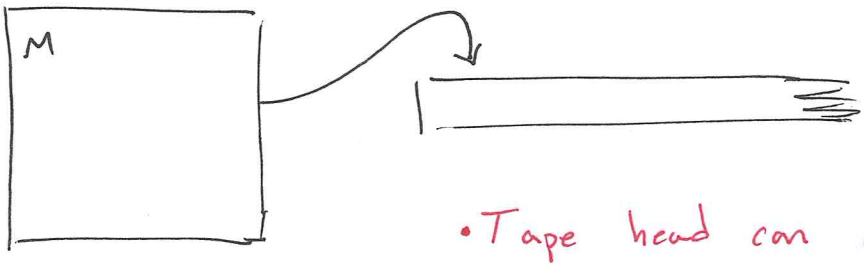
- Tape head can remain stationary
- computation can fork

PDA



- Tape head moves only to the right
- can use a stack as extra space

TM



- Tape head can move left or right.

Turing Machine Formal Definition

A TM M is a 7-tuple

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$$

Q is a finite set of states

Σ is a finite alphabet and $\sqcup \notin \Sigma$ (input alphabet)

Γ is a finite alphabet (tape alphabet)

$\sqcup \in \Gamma \qquad \Sigma \in \Gamma$

$q_0, q_{accept}, q_{reject} \in Q$

$q_{accept} \neq q_{reject}$

$$\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{Left, Right\}$$

- Q: current state
- Γ: current symbol at tape head
- Q: destination state
- Γ: character to be written
- $\{Left, Right\}$: movement

When a branch of computation reaches q_{accept} or q_{reject}, computation halts immediatly.

Can a computation never reach q_{accept} or q_{reject}?

Yes, a loop can occur

TM M_m for $C = \{ a^i b^j c^k \mid i \times j = k \text{ and } i, j, k \geq 0 \}$

M_3 = "On input string w

1. Scan input left to right to ensure it is of the form $a^+ b^+ c^+$ and REJECT if it isn't
2. Return head to beginning of tape
3. Cross off an a and scan right until a b occurs. Shuttle between b's and c's crossing one of each off until all b's are gone. If all c's are crossed out and some b's remain REJECT.
4. Restore the crossed off b's and repeat stage 3 if there are more a's to cross off. If all a's have been crossed off, check if all c's have also been crossed off. If yes ACCEPT, otherwise REJECT.

Example 2

A TM M that decides $A = \{ 0^{2^n} \mid n \geq 0 \}$

$M =$ "On input string w:

1. Sweep left to right across the tape crossing off every other 0
2. If in stage 1 the tape contained only a single 0, ACCEPT
3. If in stage 1 the tape contained an odd number of 0's greater than one REJECT.
4. Return head to beginning of Tape.
5. go to stage 1

At each iteration stage 1 cuts the number of 0's in half.

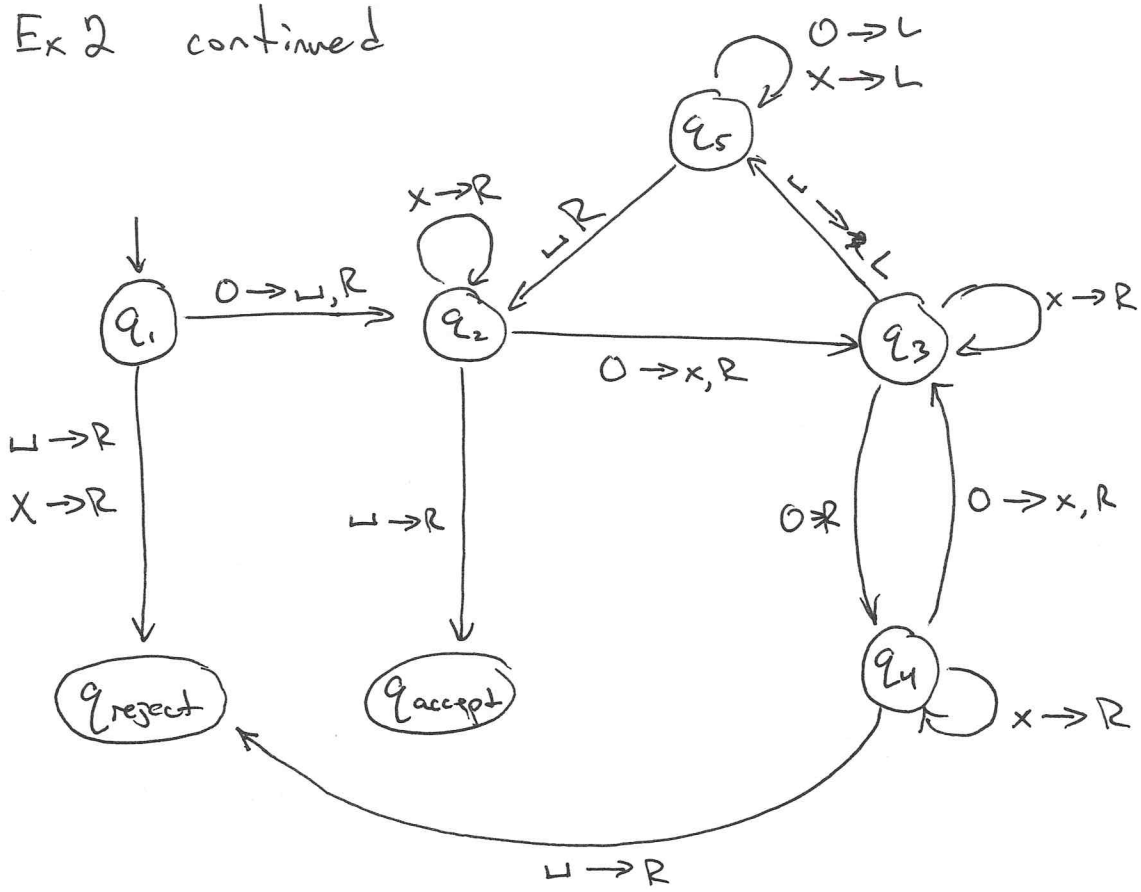
$Q = \{ q_1, q_2, q_3, q_4, q_5, q_{accept}, q_{reject} \}$

$\Sigma = \{ 0 \}$

$\Gamma = \{ 0, X, \sqcup \}$

$q_0 = q_1$

Ex 2 continued

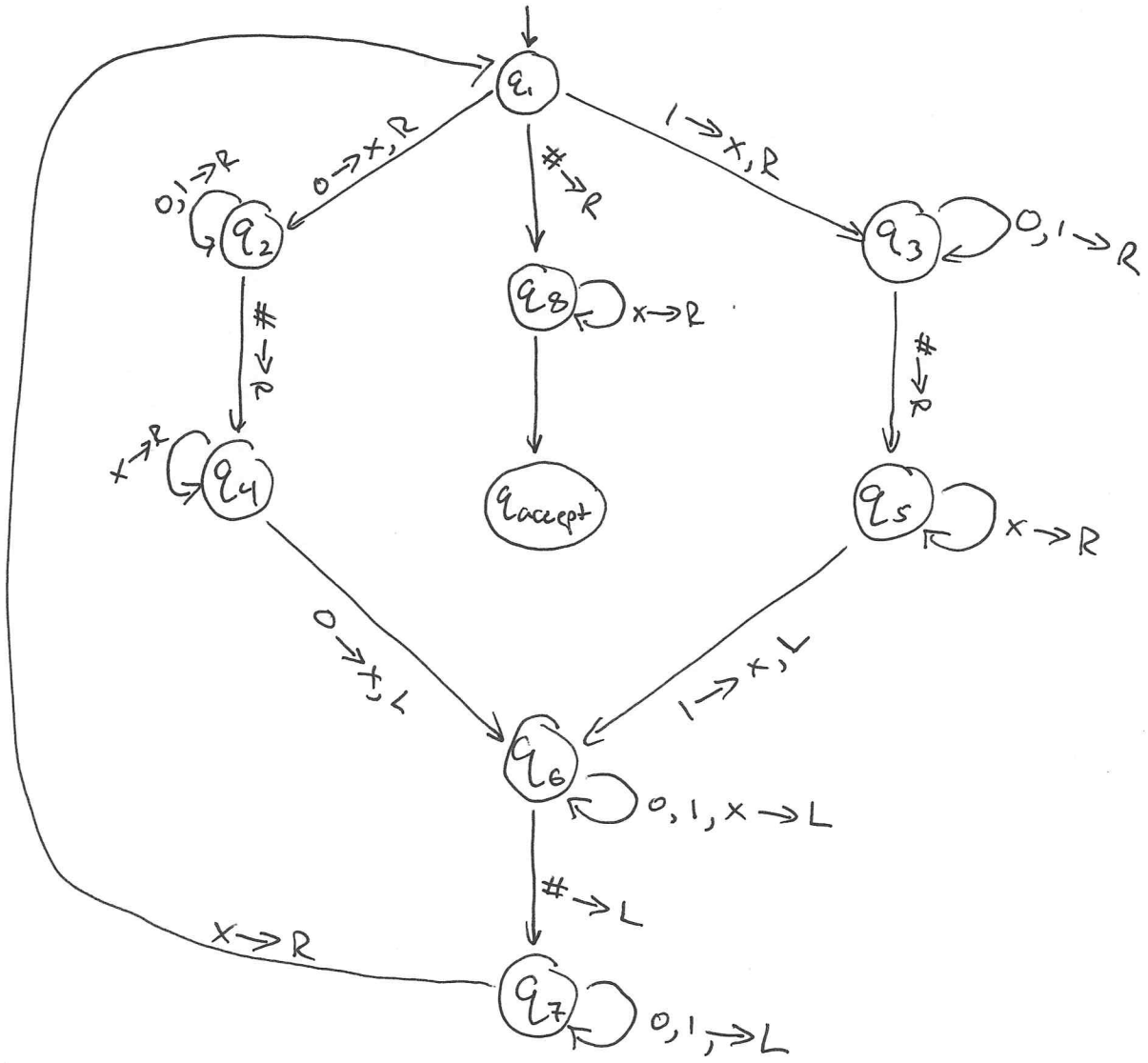


$$B = \{ w \# w \mid w \in \{0,1\}^* \}$$

Define TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$

$Q = \{ q_1, q_2 \cdots q_8, q_{accept}, q_{reject} \}$

$\Sigma = \{0, 1, \#\}$ $\quad \Gamma = \{ \sqcup, 0, 1, \#, x \}$



Decidable vs. Recognizable

- A language is Turing-Recognizable if some TM recognizes it.
  (recursively enumerable language)
- A language is Turing-Decidable if some TM decides it.
  (a recursive language)

- Is every Turing-Decidable language also Turing-Recognizable? Yes

A language is Turing-recognizable iff some NTM recognizes it.

What does it mean for a NTM to recognize a language?

A language is Turing-Decidable iff some NTM decides it.

What does it mean for a NTM to decide a language?

Many other models of general purpose computation have been proposed. Some are similar to TM and some are not. All have unrestricted access to unlimited memory.

All models with this feature are equivalent in power.

(only a finite amount of work at each step)

Multi-tape Turing Machine

$$\delta: Q \times \Gamma^k \longrightarrow Q \times \Gamma^k \times \{L, R, S\}^k$$

with k - tapes

Theorem

Every multi-tape TM has an equivalent single tape TM

S = "On input $w = w_1 \dots w_n$

1. First S puts its tape into the format that represents all k tapes of M

$$\# \dot{w}_1 w_2 \dots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \dots \#$$

2. To simulate a single move, S scans its tape from the first #, which marks the left end, to the $(k+1)$ # (right-hand end), in order to determine the symbols under the virtual heads. Then S makes a second pass to update the tapes according to how M dictates
3. If at any point S moves a virtual heads to the right onto a #. S writes a $\sqcup$ to that cell and moves all following entries on the tape to the right. "

Non-deterministic Turing Machines

$$\delta : Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

Theorem

Every nondeterministic TM has an equivalent deterministic.

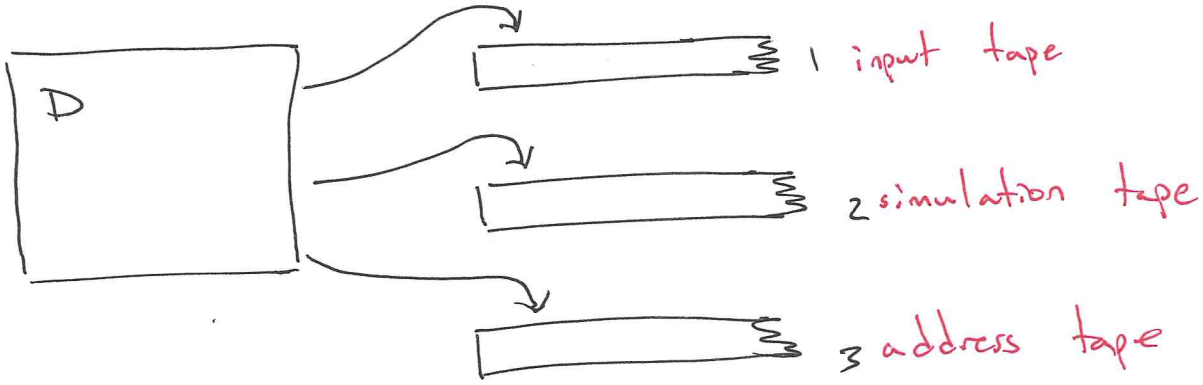
Idea

Simulate a NTM N with a TM D. If D ever finds the accept state on a branch D accepts ~~aceptst~~

Can view computation on N as a tree.

D must search all nodes. How to avoid non-terminating branches.

Breadth first search



1 input tape - never altered

2 copy of N's input tape on some branch of computation

3 where we are in the tree

1. Initially tape 1 contains the input w, and tape 2 and 3 are empty.
2. Copy tape 1 to tape 2 ~~and initialize the string on tape 3 to be ε~~
3. Use tape 2 to simulate N with input w on one branch of its non-deterministic computation. Before each step in N, consult the next symbol on tape 3 to decide which decision to make. If no more symbols remain on tape 3, or the choice is invalid, go to stage 4. Also go to stage 4 if a reject configuration is encountered. If an accepting configuration is encountered ACCEPT with 1
4. Replace the string on tape 3 with the next string in the ordering. Go to stage 2.

Description of TM

1. Formal description → explicitly defined states and transition function
2. ~~informal description~~ (implementation description)
   → English description of how the machine will move the tape heads and process input
3. high-level description → English prose describing the algorithm, ignoring the inner working of the machine.

The encoding of an object O into its representation as a string is denoted $\langle O \rangle$.
Multiple objects → $\langle O_1, O_2 \dots O_k \rangle$

What is this notation for?

Allows us to represent complex objects as strings while ignoring the internal representation.

Describe a TM M that decides.

$$A = \{ \langle G \rangle \mid G \text{ is a connected undirected graph} \}$$

M = "On input $\langle G \rangle$, (the encoded graph G)

1. Select the first node of G and mark it
2. Repeat the following stage until no new nodes are marked
3. For each node in G mark it if it is attached by an edge in G that is already marked.
4. Scan all nodes in G to determine if they are all marked. If yes ACCEPT otherwise REJECT."

$A_{DFA} = \{ \langle B, w \rangle \mid B \text{ is a DFA that accepts } w \}$

Theorem

A_{DFA} is decidable

M = "On input $\langle B, w \rangle$

1. Simulate B on input w
2. If the simulation ends in a accepting state Accept otherwise REJECT

$A_{NFA} = \{ \langle B, w \rangle \mid B \text{ is an NFA that accepts } w \}$

Theorem

A_{NFA} is decidable

N = "On input $\langle B, w \rangle$

1. Convert NFA B to DFA C
2. Run TM M on input $\langle C, w \rangle$
3. If M accepts ACCEPT, otherwise REJECT

$E_{DFA} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$

T = "On input $\langle A \rangle$

1. Mark the start state of A
2. Repeat until no new states are marked
   - Mark any state that has a transition coming into it from a marked state.
3. If no accept state is marked ACCEPT otherwise REJECT "

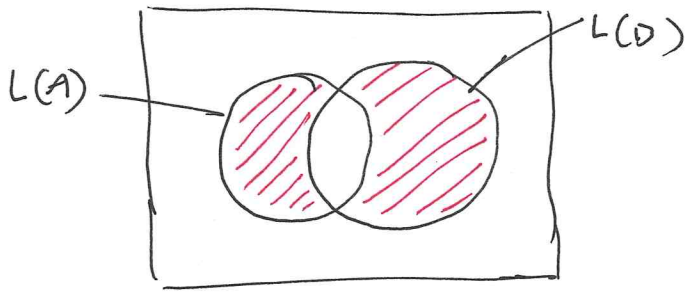
$EQ_{DFA} = \{\langle A, B\rangle \mid A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$

EQ_{DFA} is decidable

F = "On input $\langle A, B\rangle$

1. construct DFA C

$$L(C) = \left(L(A) \cap \overline{L(B)}\right) \cup \left(\overline{L(A)} \cap L(B)\right)$$



2. Run TM T on input $\langle C\rangle$
3. If T accepts ACCEPT, if T rejects REJECT

$A_{CFG} = \{\langle G, w\rangle \mid G \text{ is a CFG that generates } w\}$

is decidable

S = "On input $\langle G, w\rangle$

1. Convert G to an equivalent grammar in Chomsky Normal Form.
2. List all derivations with $2n - 1$ steps, where $n = |w|$. If $n = 0$ list all derivations with one step.
3. If any derivation generates w ACCEPT otherwise REJECT."

$E_{CFG} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$

is decidable.

R = "On input $\langle G \rangle$

1. mark ~~the start~~ all terminals in G
2. Repeat until no new variables are marked.
   - Mark any variable A where G has a rule $A \rightarrow U_1 U_2 \ldots U_k$ and each symbol $U_1, U_2 \ldots U_k$ has already been marked.
3. If the start variable is not marked ACCEPT otherwise REJECT.

Closure properties of Decidable languages

- Closed under $\cup$, $\cap$, $\circ$, $*$

Theorem

The class of Turing-Decidable languages are closed under union

if A, B are Turing Decidable then $A \cup B$ is decidable

Proof.

- Let M_1 be a decider for A and M_2 be a decider for B
- Construct TM M to decide $A \cup B$

M = "On input w:

1. Simulate M_1 on input w
   if M_1 accepts ACCEPT
2. Simulate M_2 on input w
   if M_2 accepts ACCEPT

Otherwise REJECT.