

The Mathematics of Computers

David Lloyd

d.j.lloyd@surrey.ac.uk

Contents

- **History of computers**
- **Information systems**
- **Logic theory**
- **Turing Machines**
- **The Halting Problem**
- **Mathematical Philosophy**

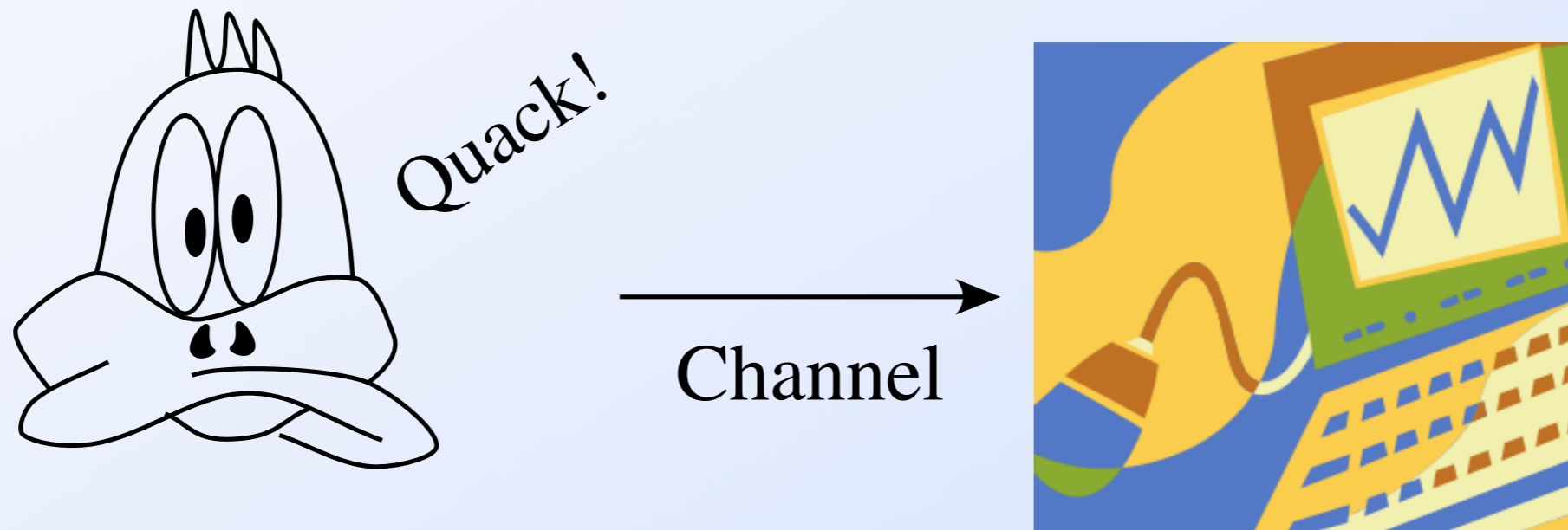
- **Abacus (2400 BC) - thought to be invented in Babylon**
- **A number of analogue computers dedicated to astronomy (150BC -)**
- **Babbage machine (1833) - bit of a failure by all accounts...**
- **Turing machines/Colossus (1936) - Alan Turing “Father of Computing”**
- **ENIAC and EDVAC (1945) - designed by John Von Neumann**

There were a loads of computers built and designed during the second world war - **mathematicians played a leading role**

- **Transistor computers (1950c -)**
- **Microchips (1960 -)**

Claude Shannon (1948) - “A Mathematical Theory of Communication”

Message error checking, cryptography etc.



- 1. Check Message: Automata** \longleftrightarrow **Grammar, formal Logic**
- 2. Execute message: Turing Machine** \longleftrightarrow **Complexity Analysis**

- So far you will have seen **Propositional Logic**:

- You start with propositions (say P and Q) that are either 1 or 0 and work out if logical statements are true or false e.g

$$P \Rightarrow (P \vee Q)$$

using truth tables.

- Propositional logic is **decidable**:

- i.e., there exists an algorithm s.t. it is always possible to prove or disprove a theorem in finite time.

However, the algorithm may take exponential time to complete...

- The algorithm for determining the truth (or not) of a logical statement is called: **Proof by Resolution**

- Computer programs for carrying out Proof by Resolution:

- Otter, Vampire, Isabelle etc.
- Logic Programming: Prolog, Fril etc.

- How do I check the validity of the following arguments?

All scientists have silly hair,
Einstein was a scientist,
Therefore Einstein had silly hair.

All mathematicians are kangaroos,
Dave is a mathematician,
Therefore Dave is a kangaroo :-)

- Propositional logic can't help, we need **Predicate logic**...
- Introduce variable, objects, properties and relationships (functions) e.g.,

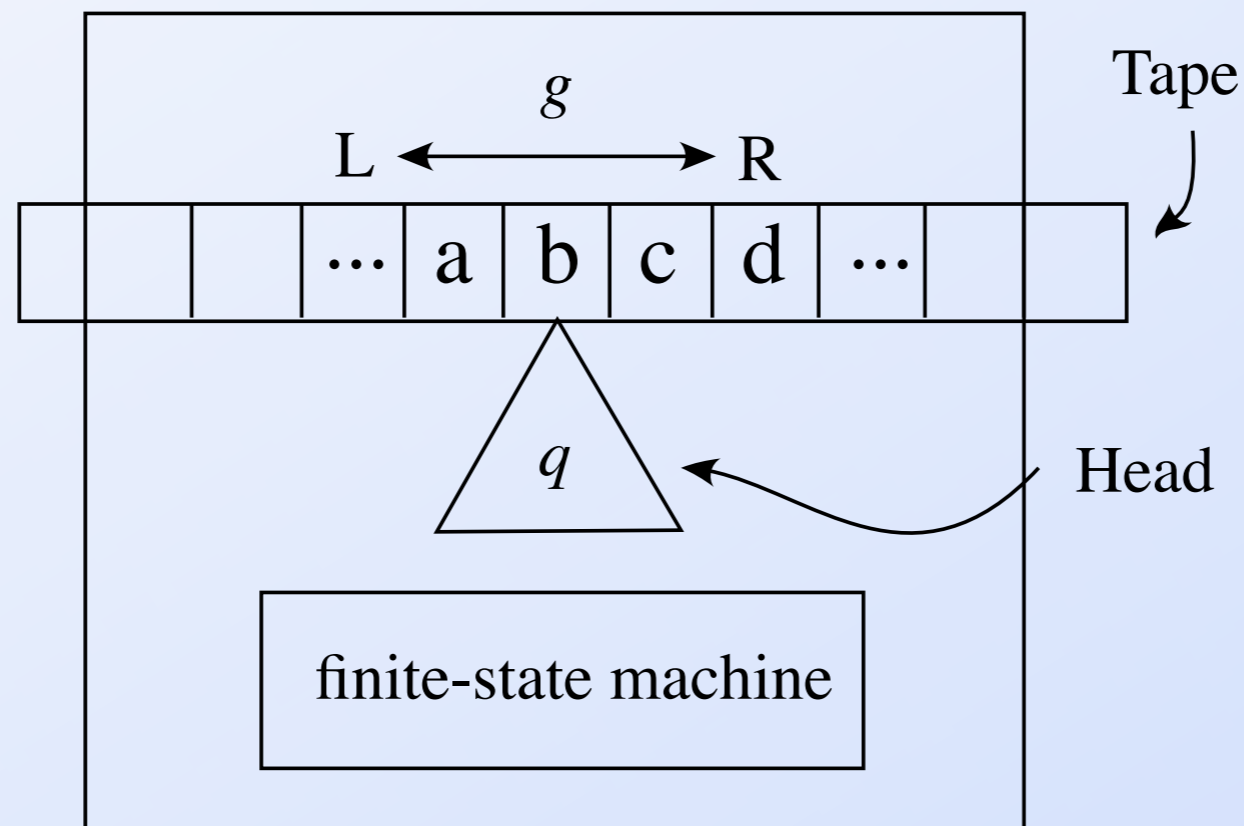
Text

$$\forall x(\text{Mathematician}(x) \rightarrow \text{Kangaroo}(x)),$$
$$\text{Mathematician}(\text{Dave}),$$
$$\Rightarrow \text{Kangaroo}(\text{Dave})$$

- Predicate logic is **semi-decidable**:
 - i.e., there exists an algorithm s.t. it is possible to demonstrate inconsistency in finite time, but the procedure may not terminate if the set of sentences is consistent.
 - Can we find a machine to determine if an algorithm will terminate?
 - Proof by Resolution example:
 - <http://www.netfunny.com/rhf/jokes/90q4/burnher.html>

Turing Machine

- Described by Alan Turing, 1936
- An abstract computer: can carry out any algorithm



Tape:

- unbounded in length
- may be read in either direction
- may be written any time

Formal definition

- Turing machine consists of (Q, T, g, q_0, F)

Q - finite set of states

q_0 - initial state

F - final state $\subseteq Q$

T - finite vocabulary of symbols

g - transition function, a **partial function** from

$$(Q \setminus F) \times T \rightarrow Q \times T \times \{L, R\}$$

Thus

$$g(q, a) = (q', a', X)$$

where $q, q' \in Q, a, a' \in T, X \in \{L, R\}$

defines a move, “L” left and “R” right of the head

Turing Machine

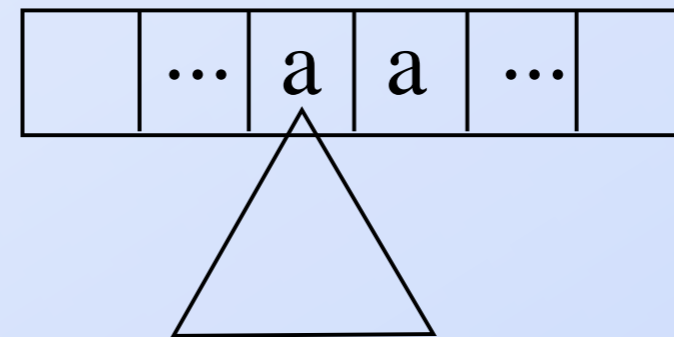
- **Action of the Turing Machine (TM):**

- initial series of symbols (from T) on tape
- The move depends on current state (q) and the symbol (a) under head
- Move consists of:
 1. a change of state (to q'),
 2. writing a new symbol onto the tape (a'),
 3. shift of the head L or R.

- **TM “halts and succeeds”** if $q \in F$

- **TM “halts and fails”** if $q \in Q \setminus F$ and no move possible

- **Endless loop:**



$$g(q_1, a) = (q_2, a, R),$$

$$g(q_2, a) = (q_1, a, L), \quad \text{where } q_1, q_2 \notin F$$

Turing machine

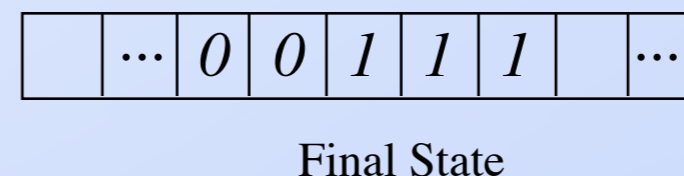
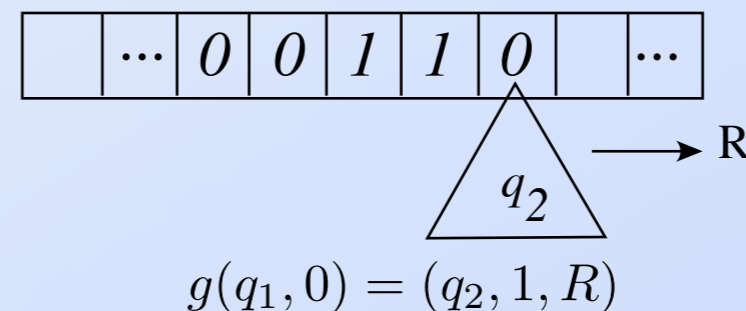
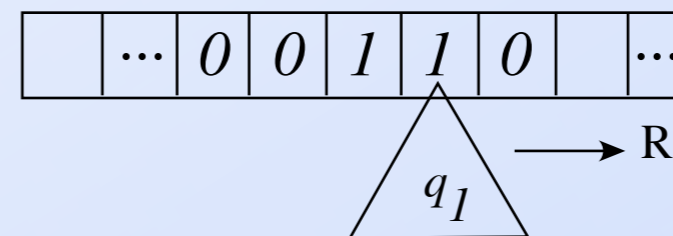
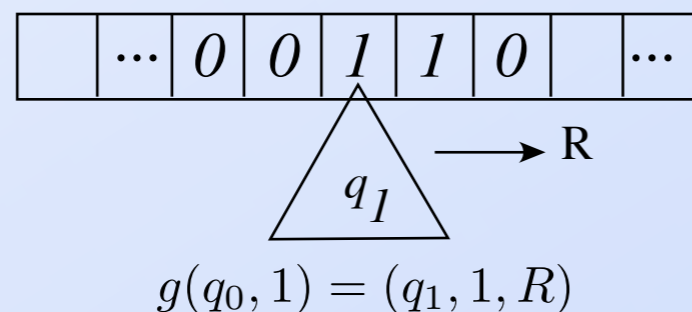
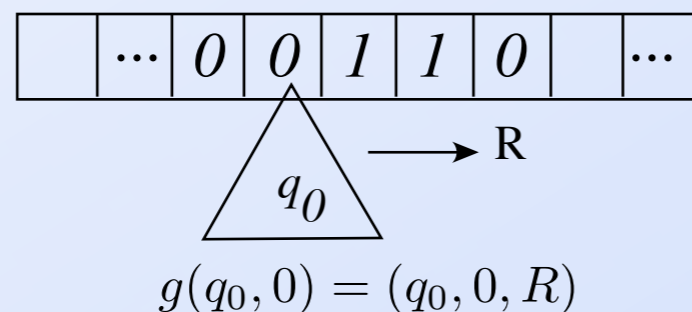
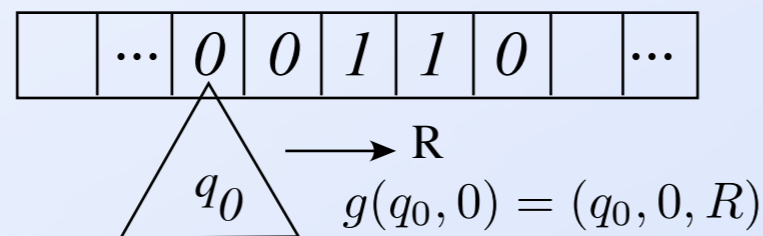
- An initial state is “accepted” by a TM if it halts and succeeds.
- If a TM machine always halts for **any** input then it defines an “algorithm”

Example: Move right skipping over zeros.

If head meets a sequence of ones, add one to the end of the sequence, halt and succeed.

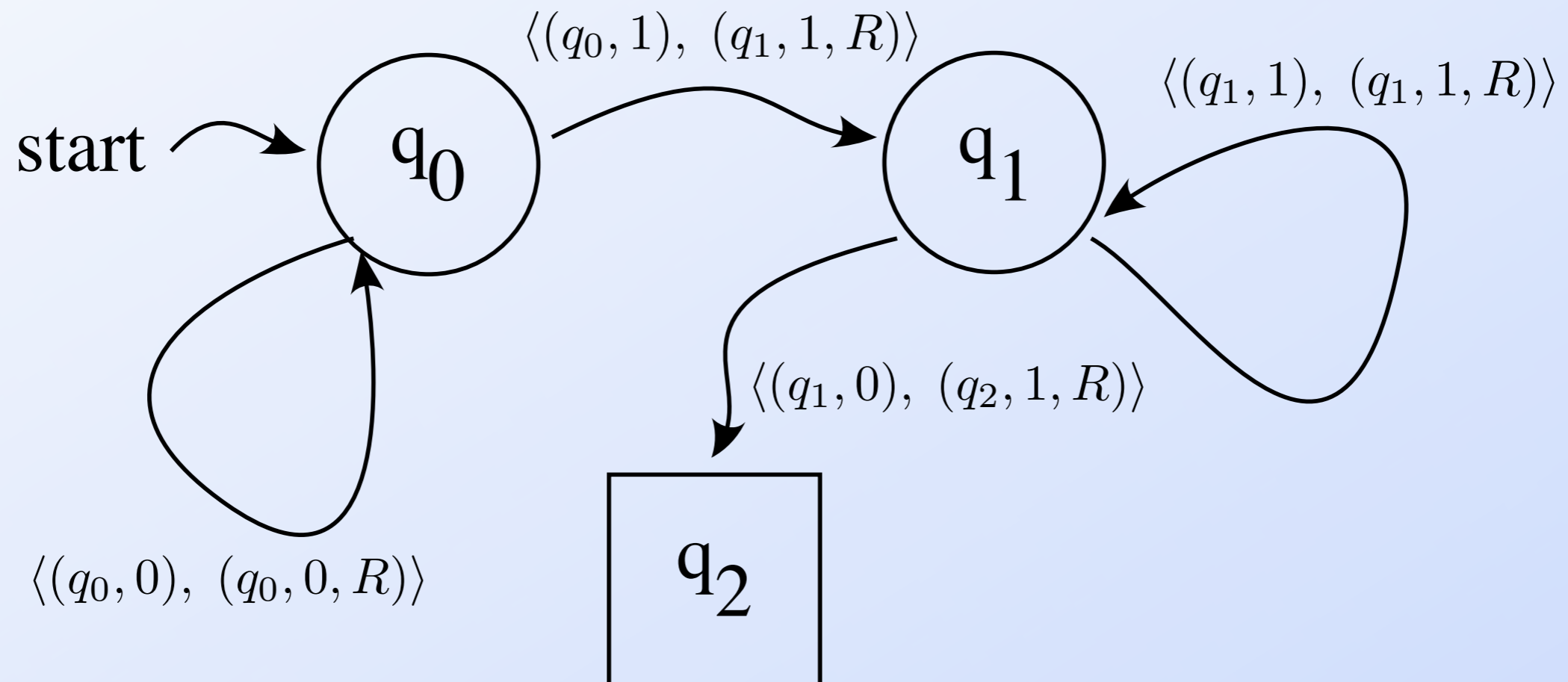
$$Q = (q_0, q_1, q_2), \quad T = \{0, 1\}, \quad F = \{q_2\},$$

$$g(q_0, 0) = (q_0, 0, R), \quad g(q_1, 0) = (q_2, 1, R), \quad g(q_0, 1) = (q_1, 1, R), \quad g(q_1, 1) = (q_1, 1, R)$$



State diagrams

- **Another way to visualise a Turing machine**



- **Turing machines may not always halt!**

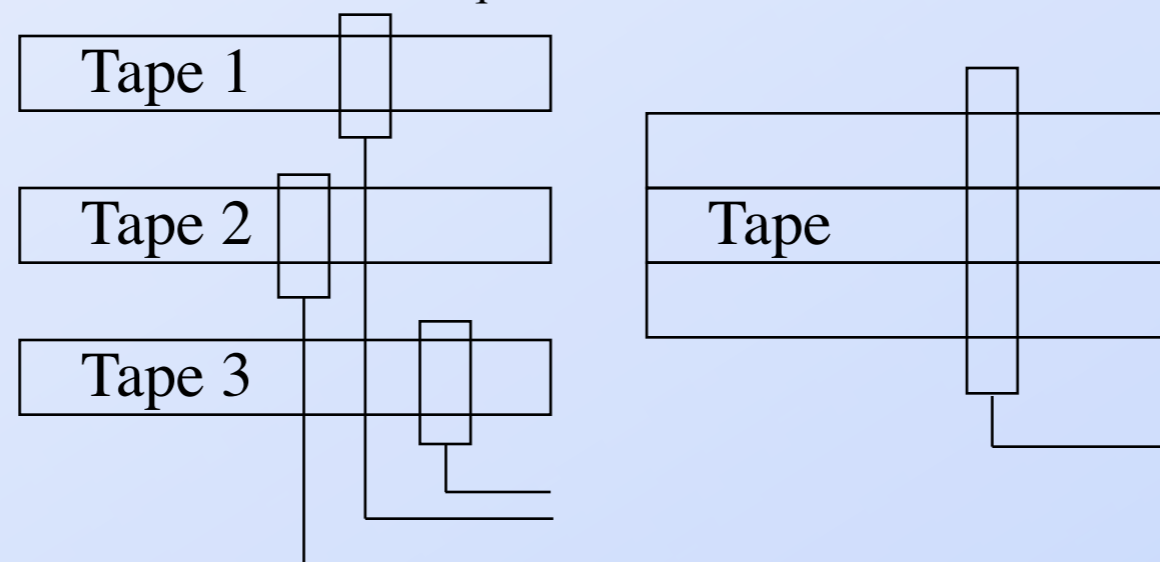
Extensions to the TM

- **Possible extensions to the simple TM**
 - **two-way infinite tape**
 - **multiple tracks on the tape**
 - **multiple heads on the tape**
 - **multiple tapes, etc...**
- **A TM with any/all of these extensions is equivalent to a simple TM**
 - **i.e., they give no extra “power”!**

But they may compute faster...

Example: For a multi-tape and multi-track machine

Equivalent TMs



Turing machine: Example

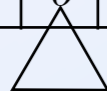
- **TM: add two binary numbers and leave the sum on the tape.**

The tape has two tracks, each initial containing one number - \$ symbol as left end mark

The total is left on one track with the other one blank (b)

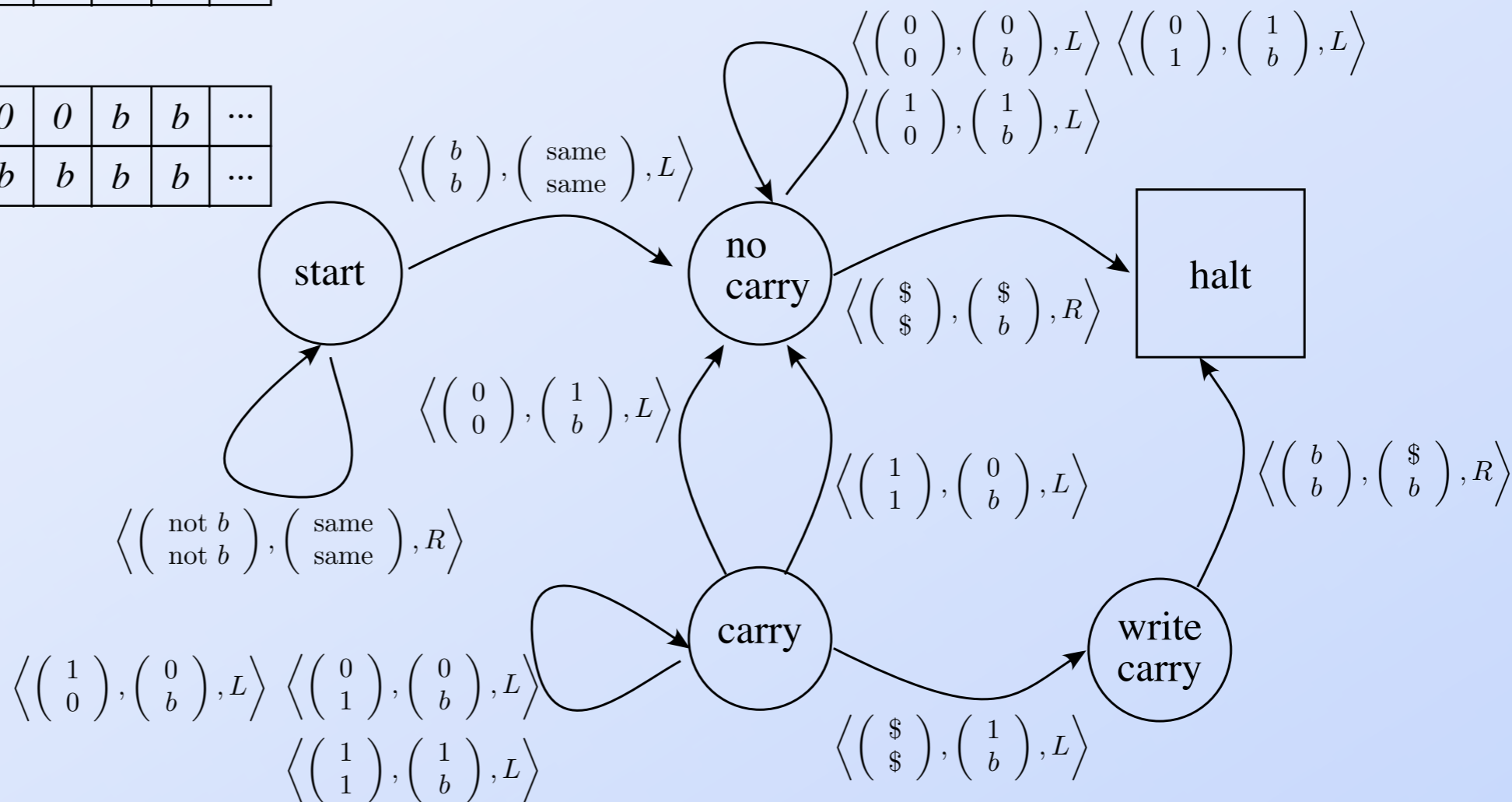
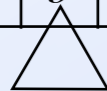
Start

...	b	\$	0	1	0	1	1	b	b	...
...	b	\$	0	1	1	0	1	b	b	...



Finish

...	b	\$	1	1	0	0	0	b	b	...
...	b	b	b	b	b	b	b	b	b	...



Universal Turing Machine

- **Input tape to a TM can contain the program of the TM**
- **Universal Turing Machine (UTM):**
 - **A TM that can read an arbitrary program (TM)**
 - **Then reproduce the same output**

Input to a UTM: $e(M)oX$

$e(M)$ - suitable encoding of a TM, M

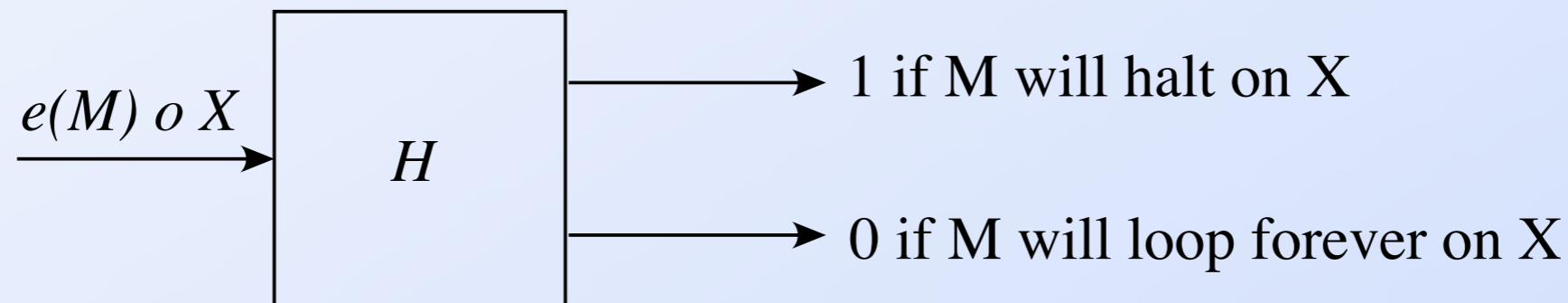
o - separator symbol

X - input to M

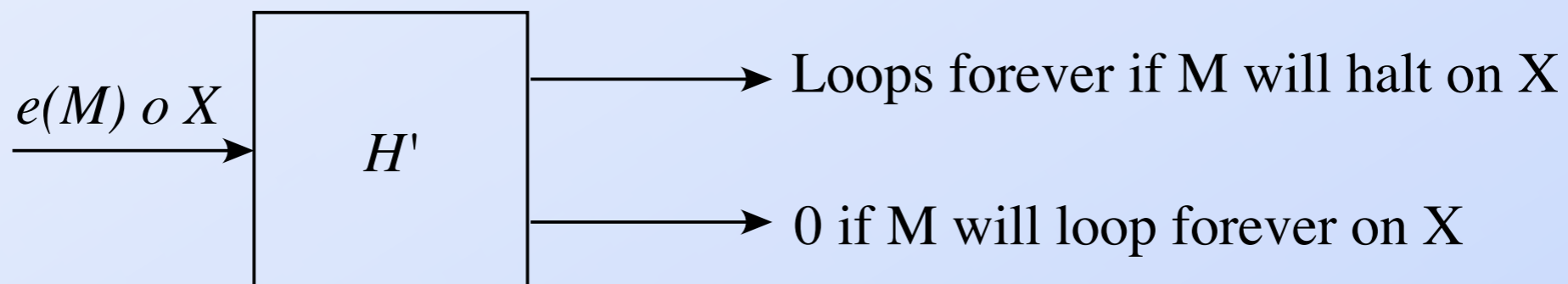
- **The smallest known UTM has 2-states and 5-symbols...**

The Halting Problem

- Can we work out if a TM will halt or not?
- Suppose that M may loop forever on input X
- Try and construct a UTM (H) input $e(M) \circ X$
 - output 1 if M eventually halts
 - output 0 if M loops forever

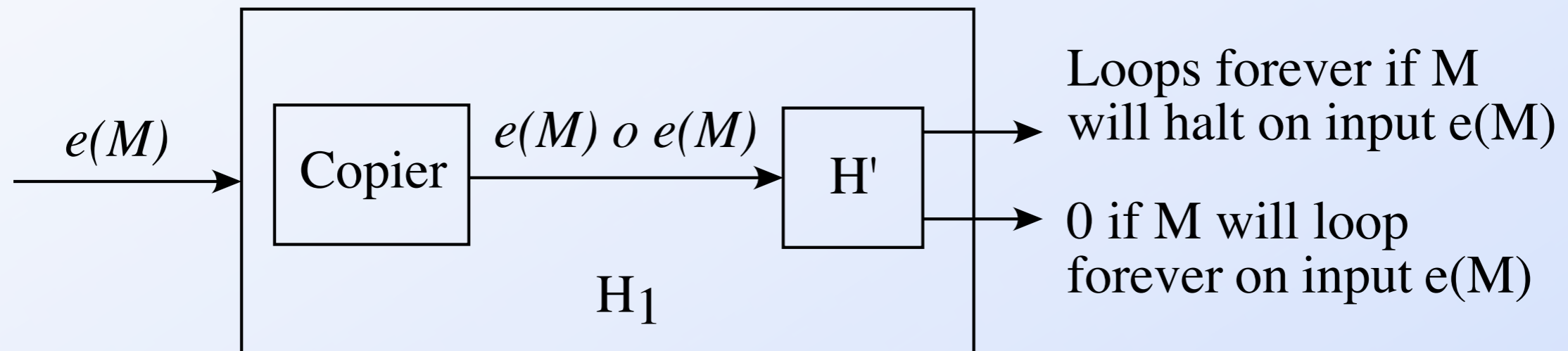


- A minor variation of H (say H') loops forever if M will halt on input X :



The Halting Problem

- Now construct H_1 which first copies its input then feeds it to H'



- Suppose we now input $e(H_1)$ to H_1 , then:
 - H_1 loops forever if H_1 will halt on $e(H_1)$
 - H_1 halts (with output 0) if H_1 will loop forever on input $e(H_1)$
- **Contradiction** \Rightarrow So H_1 can not exist
 - The Copier is no problem so H' can not exist
 - Therefore H can not exist

No TM can decide whether or not an arbitrary TM will eventually halt when presented with an arbitrary input X

- **By the Church-Turing thesis:**

- **it is not effectively computable whether an arb. TM will halt or loop forever when presented with an arb. input X.**

This is worse than intractable: the problem is not solvable even in principal!

- **So how does one write programs for program verification, software reliability testing?**

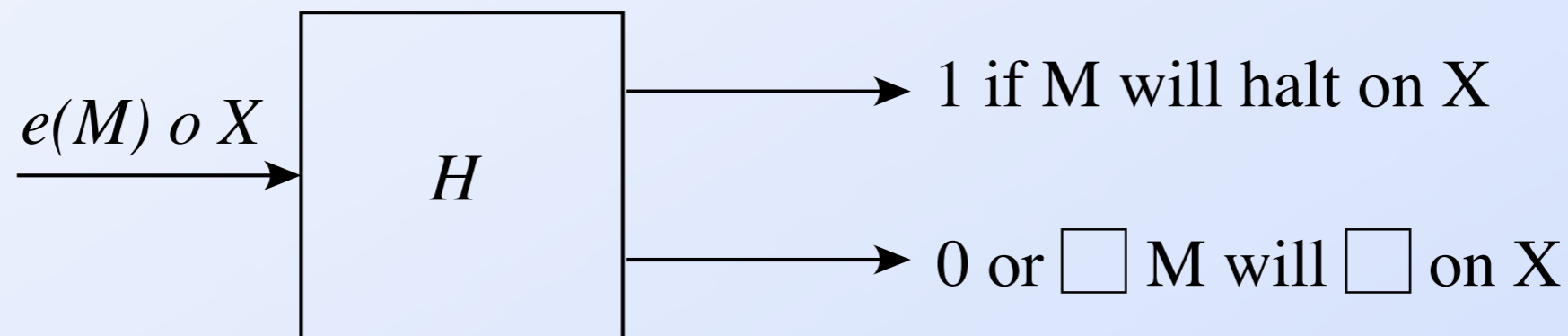
Hilbert's programme (early 1900s):

- **Mathematics is **consistent**: it is not possible to prove both a statement and its opposite**
- **Mathematics is **complete**: every true mathematical statement is provable**
- **Mathematics is **decidable**: there is a formal procedure s.t. for every mathematical statement it can be shown whether or not the statement is true.**

**Gödel (1930) proved that (1) and (2) cannot both be true in formal arithmetic
Church (1936) showed that Predicate logic was undecidable.**

A final twist

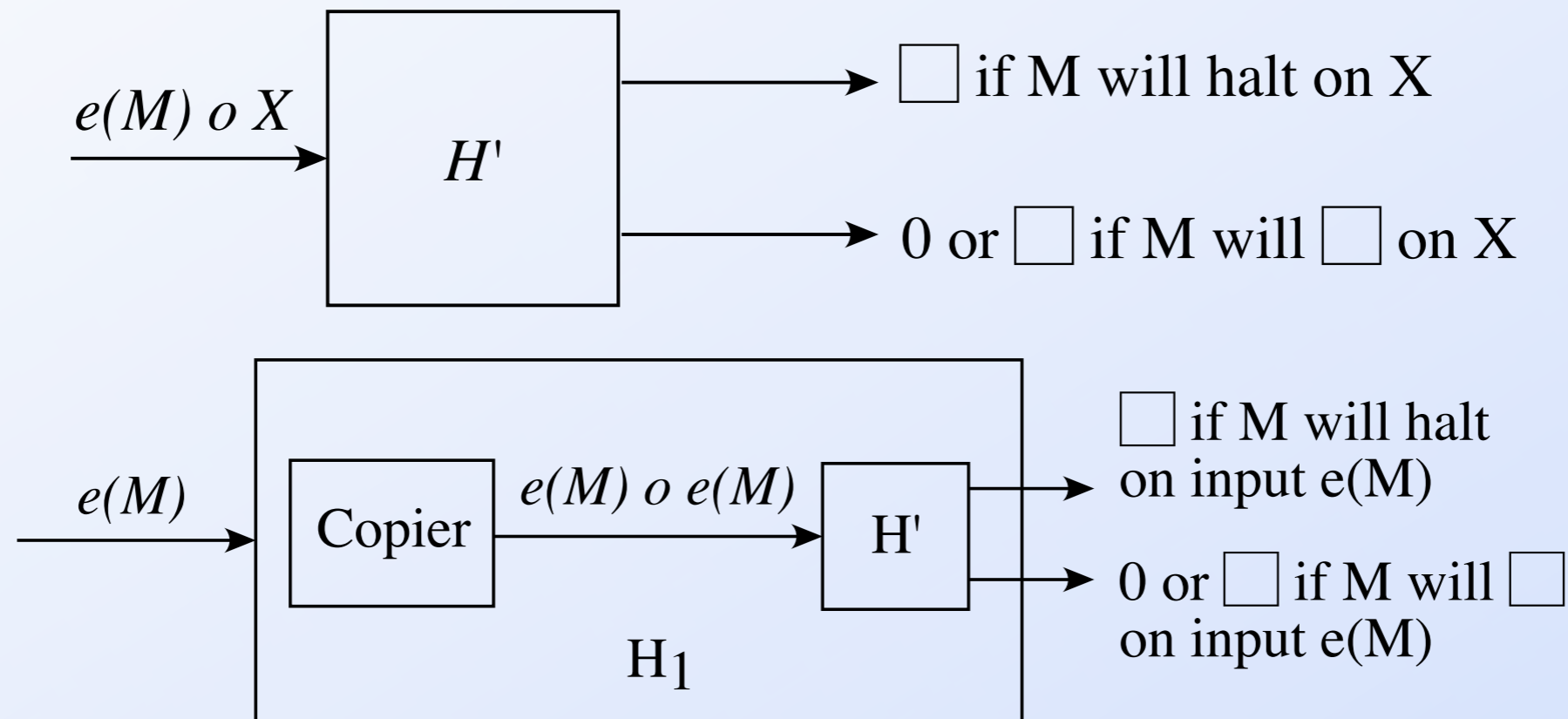
- One further implication of the halting problem...
- Since **H** can't exist as specified before, modify def. **H** outputs either
 - **0**, or
 - **loops forever if M will loop forever on input X**



The symbol \square denotes “loops forever”.

- Now, create H' and H_1 as before...

A final twist



- **Suppose we input $e(H_1)$ to H_1 :**
 - **Assume H_1 will halt \Rightarrow response is \square . Contradiction**
 - **Assume response is $\square \Rightarrow$ no contradiction, H_1 can exist but not output 0**
 - **$\Rightarrow H'$ (and correspondingly H) cannot output 0 but \square**

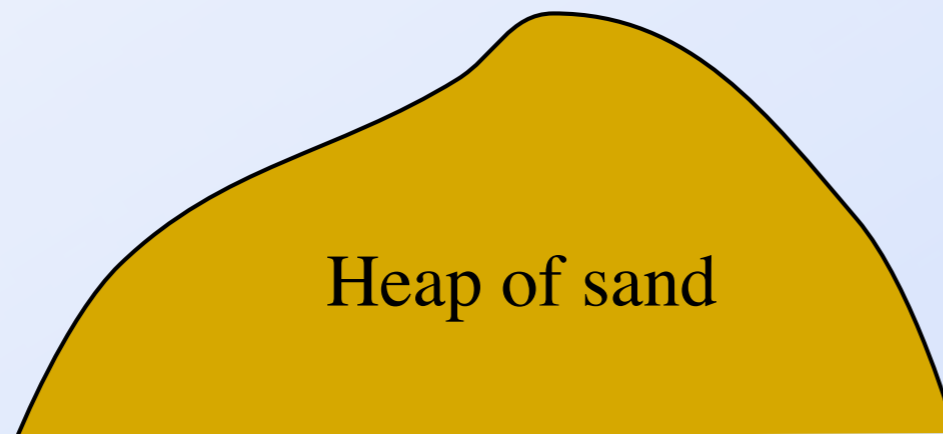
So H fails to give an answer if we attempt to use it to determine whether or not H_1 will halt.

A final twist

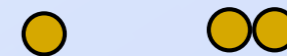
- So H_1 will not halt and defeats H (yet we know the answer!)
- Given **any** proposed algorithm for deciding whether or not a TM halts, we can produce a TM that **we** can see does not halt, yet the algorithm cannot.
- Conclusion(?): Humans must embody processes that cannot be described by any algorithm
 - (see R. Penrose, “The Emperor’s New Mind”, Oxford Press, 1989)
- Paradox(?): The argument on previous slide is clearly rational and well-defined...
- But we are clearly using some sort of calculation...
- Conscience...

“I want you to believe the things I tell you, but **not** because you **believe me**; I want you, rather, to believe them because **you** yourself see that they must be true”

- Reasoning with uncertainty, e.g. chess programs,
 - They don't calculate checkmate,
 - they play the “**best**” legal chess move
- How to decide what the “best” move is?
- Q: When do I have a “heap” of sand?



Not a Heap of sand?



Implications for abortion law?