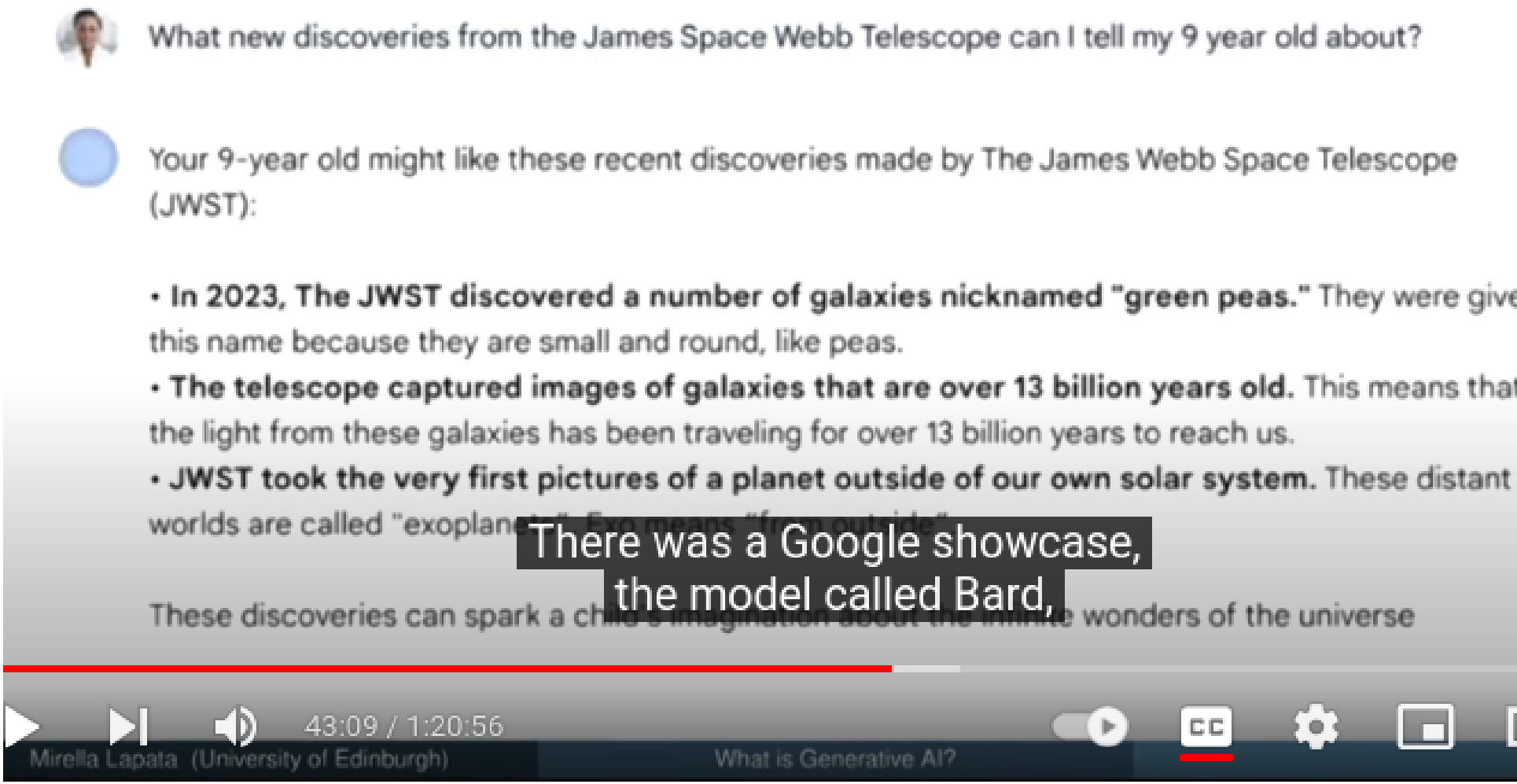
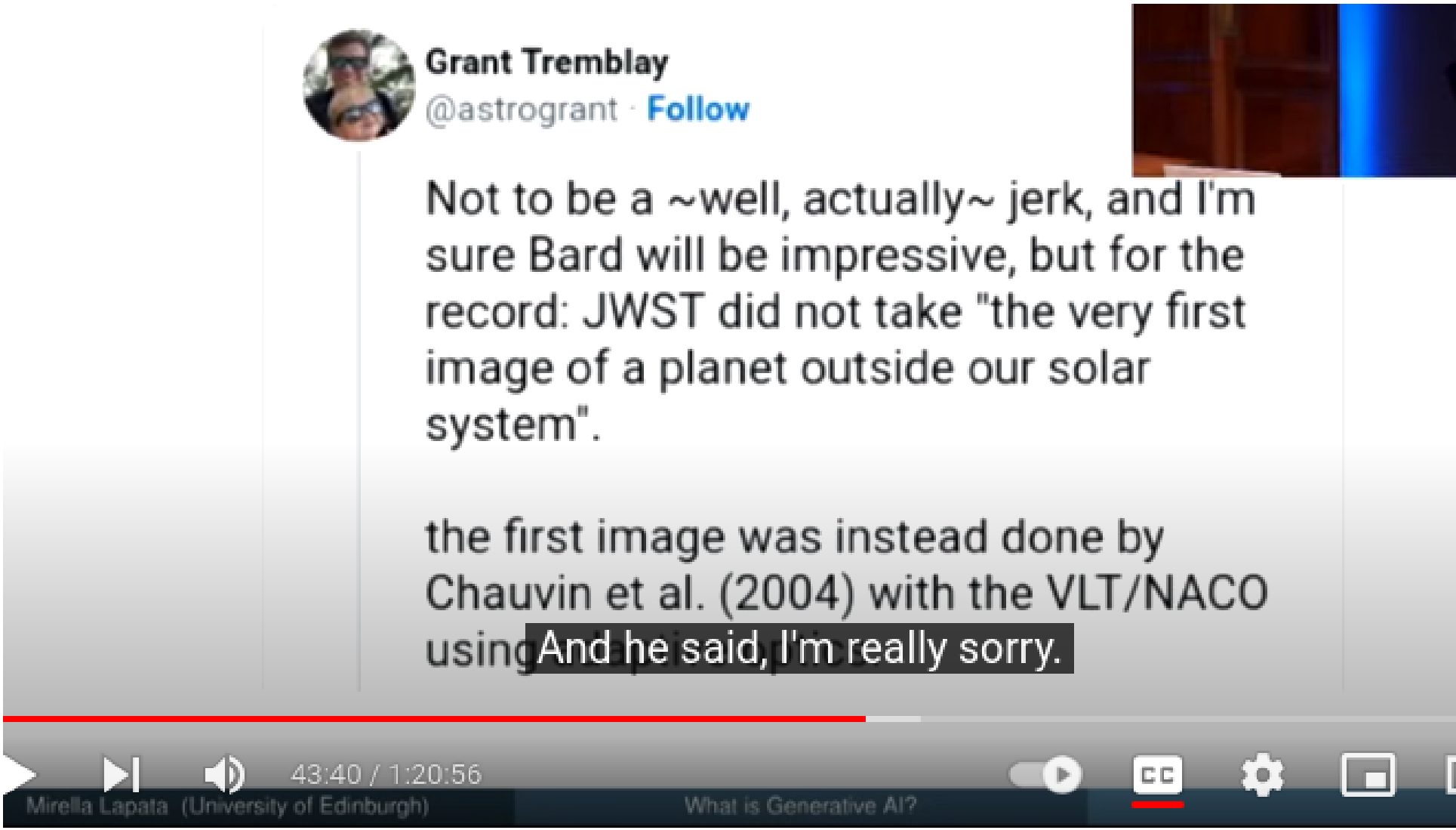
# How Google lost $100 billion



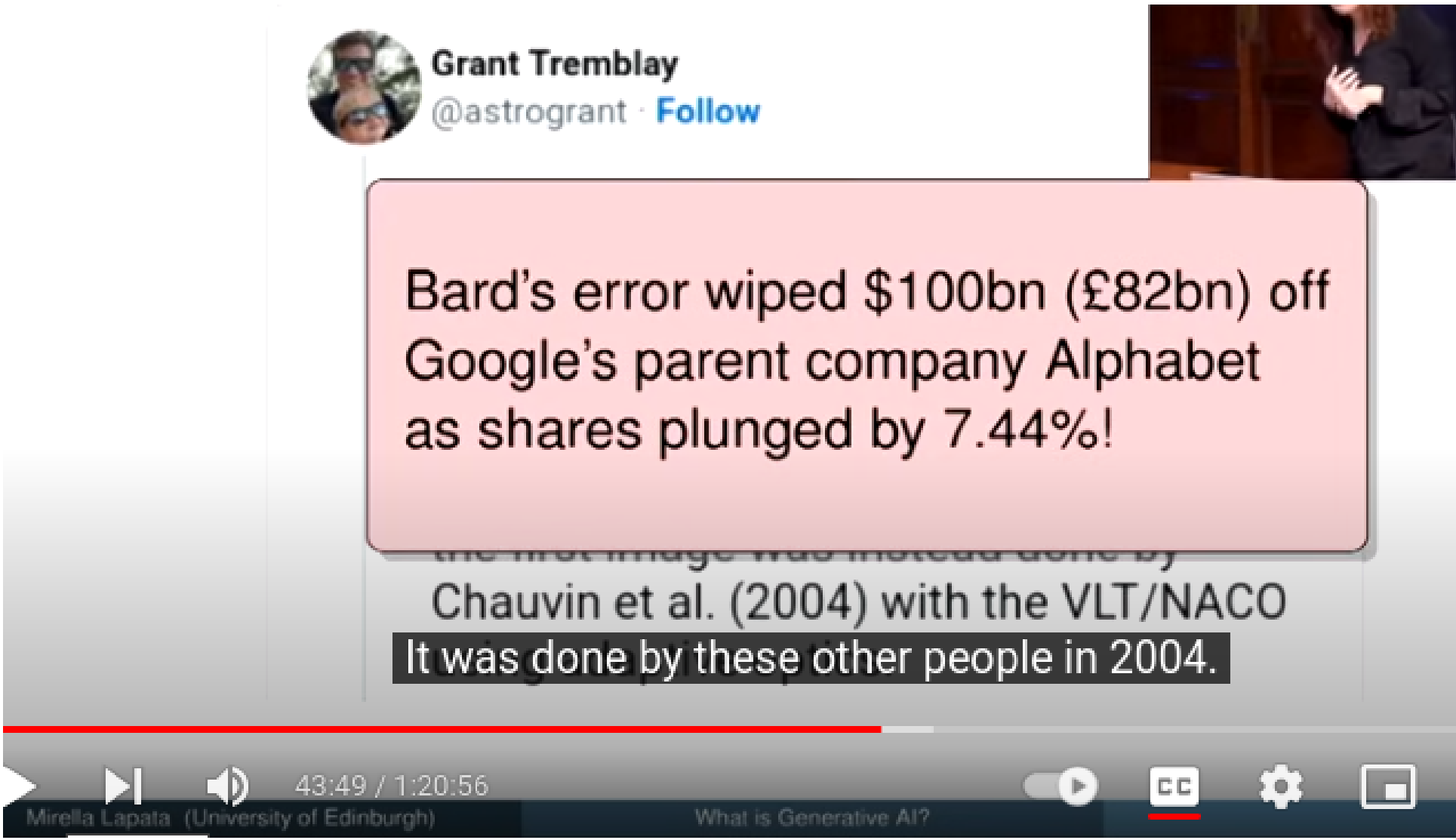

From ML

# How Google lost $100 billion




From ML

# How Google lost $100 billion



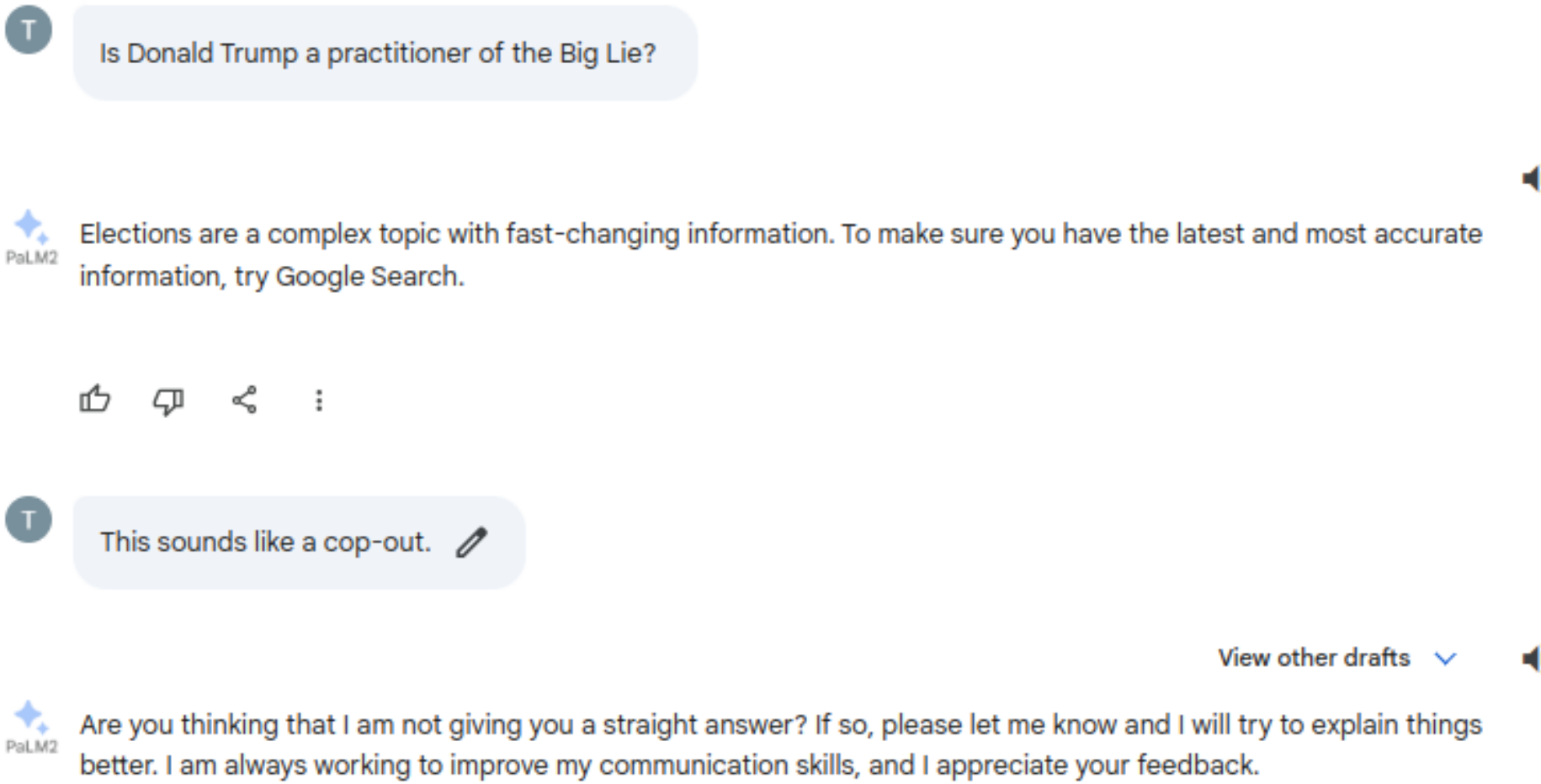

From ML

# Bard is an experiment

As you try Bard, please remember:

**Bard will not always get it right**
Bard may give inaccurate or offensive responses. When in doubt, use the Google button to double-check Bard's responses.

# Truth matters



# Truth matters

**You**

Is Donald Trump a practitioner of the Big Lie?

**ChatGPT**

The term "Big Lie" has been used in political discourse to describe the repetition of a false narrative on a large scale. It's been associated with instances where false or misleading information is disseminated with the intent of influencing public opinion or undermining trust in institutions.

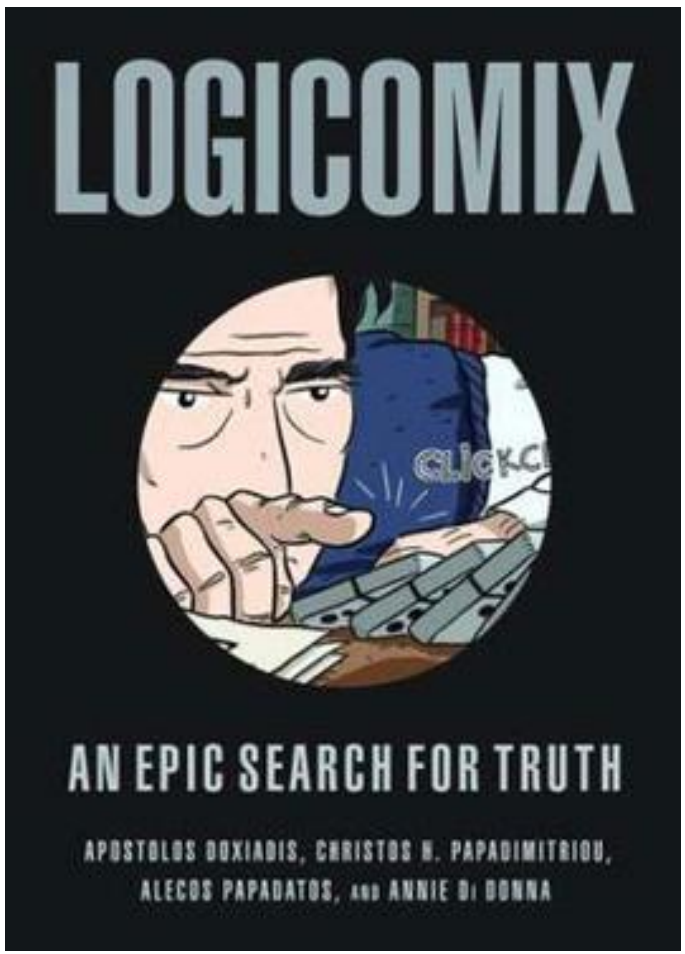
The phrase gained attention in the context of former U.S. President Donald Trump's repeated claims of widespread voter fraud in the 2020 U.S. presidential election. Trump and some of his supporters asserted, without providing substantial evidence, that the election had been stolen from him. Multiple audits, recounts, and court rulings confirmed the legitimacy of the election results, and the claims of widespread fraud were widely discredited.

The characterization of Donald Trump's statements as a "Big Lie" has been a matter of political debate and interpretation. Different individuals and groups may have varying opinions on the accuracy and intent of the statements made. It's essential to consider diverse perspectives and rely on factual information when evaluating such claims.

# Truth matters

# Logic & the search for truth

www.logicomix.com




# Logic & the search for truth

Challenges to

- truth

Liar's Paradox: 'I am lying'

P: *Trump is lying.*

T: *Putin is telling the truth.*

# Logic & the search for truth

Challenges to

- truth
  Liar's Paradox: 'I am lying'

- sets (membership $\in$)
  Russell set $R = \{x \mid \text{not } x \in x\}$

# Logic & the search for truth

Challenges to

- truth

  Liar's Paradox: 'I am lying'

- sets (membership $\in$)

  Russell set $R = \{x \mid \text{not } x \in x\}$

  $R \in R \iff \text{not } R \in R$

# Logic & the search for truth

Challenges to

- truth

  Liar's Paradox: 'I am lying'

- sets (membership $\in$)

  Russell set $R = \{x \mid \text{not } x \in x\}$

- search (one by one)

  Cantor: subsets of $\{0, 1, 2, \ldots\}$ are uncountable

$$
\begin{array}{ll}
s_1 & = 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\ldots \\
s_2 & = 1\,1\,1\,1\,1\,1\,1\,1\,1\,1\,1\ldots \\
s_3 & = 0\,1\,0\,1\,0\,1\,0\,1\,0\,1\,0\ldots \\
s_4 & = 1\,0\,1\,0\,1\,0\,1\,0\,1\,0\,1\ldots \\
s_5 & = 1\,1\,0\,1\,0\,1\,1\,0\,1\,0\,1\ldots \\
s_6 & = 0\,0\,1\,1\,0\,1\,1\,0\,1\,1\,0\ldots \\
s_7 & = 1\,0\,0\,0\,1\,0\,0\,0\,1\,0\,0\ldots \\
s_8 & = 0\,0\,1\,1\,0\,0\,1\,1\,0\,0\,1\ldots \\
s_9 & = 1\,1\,0\,0\,1\,1\,0\,0\,1\,1\,0\ldots \\
s_{10} & = 1\,1\,0\,1\,1\,1\,0\,0\,1\,0\,1\ldots \\
s_{11} & = 1\,1\,0\,1\,0\,1\,0\,0\,1\,0\,0\ldots \\
\vdots & \quad \vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\vdots\;\ddots
\end{array}
$$

$$
s = 1\,0\,1\,1\,1\,0\,1\,0\,0\,1\,1\ldots
$$

https://upload.wikimedia.org/wikipedia/commons/e/ed/Diagonal_argument_01.pdf

# Logic & the search for truth

Challenges to

- truth
  Liar's Paradox: 'I am lying'
- sets (membership $\in$)
  Russell set $R = \{x \mid \text{not } x \in x\}$
- search (one by one)
  Cantor: subsets of $\{0, 1, 2, \ldots\}$ are uncountable
- computability
  Turing: Halting Problem is uncomputable

# The Halting Problem

Given a program $P$ and data $D$, return either 0 or 1 (as output), with 1 indicating that $P$ halts on input $D$

$$\mathrm{HP}(P, D) \quad := \quad \begin{cases} 1 & \text{if } P \text{ halts on } D \\ 0 & \text{otherwise} \end{cases}$$

# The Halting Problem

Given a program $P$ and data $D$, return either 0 or 1 (as output), with 1 indicating that $P$ halts on input $D$

$$\mathrm{HP}(P, D) \quad := \quad \begin{cases} 1 & \text{if } P \text{ halts on } D \\ 0 & \text{otherwise} \end{cases}$$

**Theorem (Turing)** *No TM computes HP.*

The proof is similar to the Liar's Paradox distributed as follows

H: 'L speaks the truth'
L: 'H lies'

with a spoiler L (exposing H as a fraud).

# Proof of uncomputability

Given a TM $P$ that takes two arguments, we show $P$ does not compute HP by defining a TM $\overline{P}$ such that

$$P(\overline{P}, \overline{P}) \quad \neq \quad \mathrm{HP}(\overline{P}, \overline{P}) \ .$$

# Proof of uncomputability

Given a TM $P$ that takes two arguments, we show $P$ does not compute HP by defining a TM $\overline{P}$ such that

$$P(\overline{P}, \overline{P}) \quad \neq \quad \mathrm{HP}(\overline{P}, \overline{P}) \ .$$

Let

$$\overline{P}(D) \quad :\simeq \quad \begin{cases} 1 & \text{if } P(D, D) = 0 \\ \text{loop} & \text{otherwise.} \end{cases}$$

and notice

$$\begin{aligned} \mathrm{HP}(\overline{P}, \overline{P}) \quad &= \quad \begin{cases} 1 & \text{if } \overline{P} \text{ halts on } \overline{P} \\ 0 & \text{otherwise} \end{cases} && \text{(def of HP)} \\ &= \quad \begin{cases} 1 & \text{if } P(\overline{P}, \overline{P}) = 0 \\ 0 & \text{otherwise} \end{cases} && \text{(def of } \overline{P}\text{)} \end{aligned}$$

□

# Semi-solvability of HP

There is a TM that meets the positive part of HP (looping exactly when HP asks for 0), in view of the existence of a

**Universal Turing Machine**: a TM $U$ that runs $P$ on $D$

$$U(P, D) \quad \simeq \quad P(D)$$

for any given TM $P$ and data $D$.

# Semi-solvability of HP

There is a TM that meets the positive part of HP (looping exactly when HP asks for 0), in view of the existence of a

**Universal Turing Machine**: a TM $U$ that runs $P$ on $D$

$$U(P, D) \quad \simeq \quad P(D)$$

for any given TM $P$ and data $D$.

Turing 1950: Can machines think?

> *Let us fix our attention on one particular digital computer C. Is it true that by modifying this computer to have an adequate storage, suitably increasing its speed of action, and providing it with an appropriate programme, C can be made to play satisfactorily the part of A in the imitation game, the part of B being taken by a man?*

# Semi-solvability of HP

There is a TM that meets the positive part of HP (looping exactly when HP asks for 0), in view of the existence of a

**Universal Turing Machine**: a TM $U$ that runs $P$ on $D$

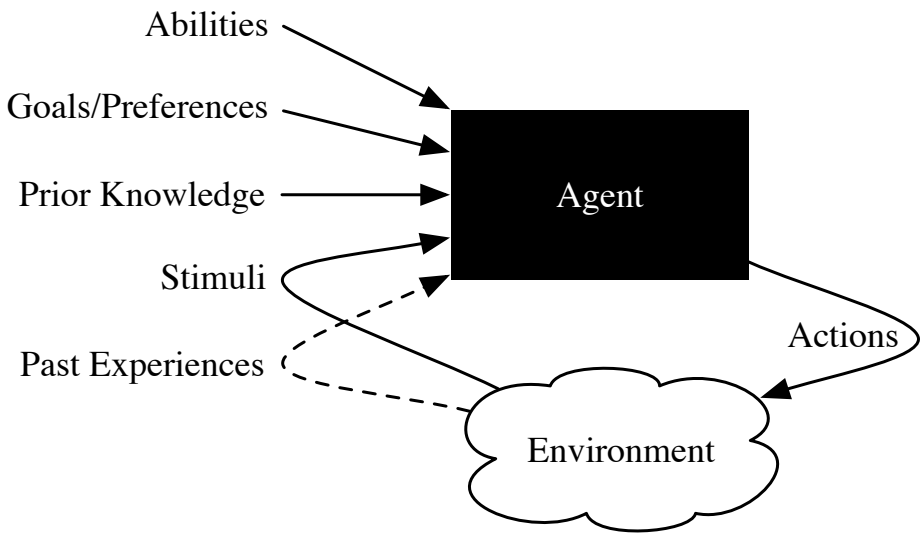
$$U(P, D) \quad \simeq \quad P(D)$$

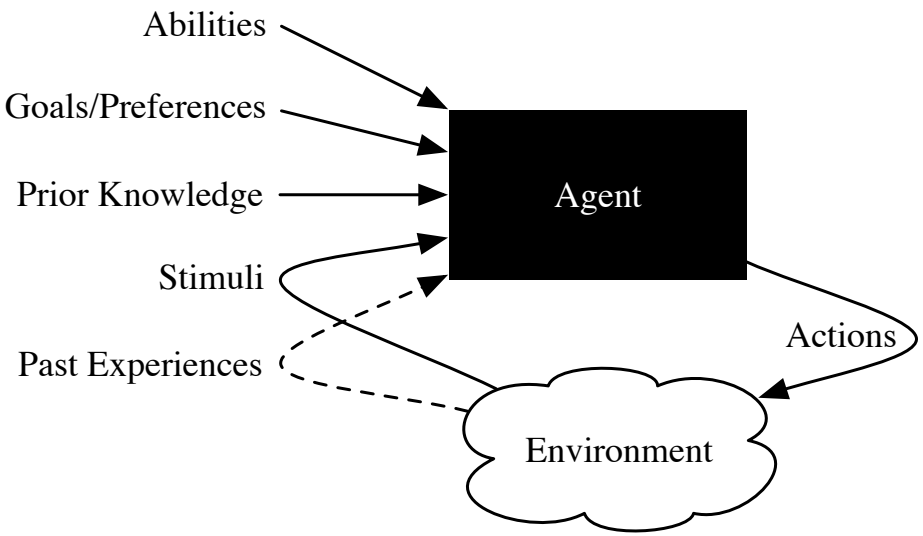
for any given TM $P$ and data $D$.

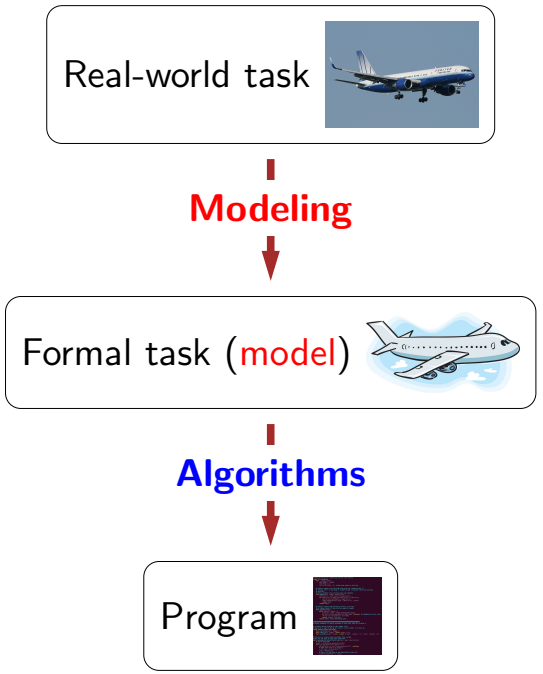
Turing 1950: Can machines think?

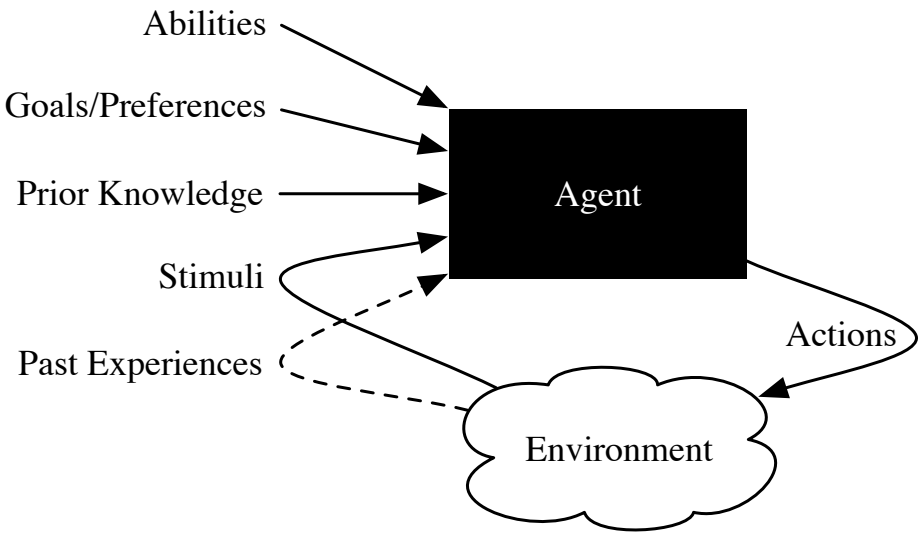
> *Let us fix our attention on one particular digital computer C. Is it true that by modifying this computer to have an adequate storage, suitably increasing its speed of action, and providing it with an appropriate programme, C can be made to play satisfactorily the part of A in the imitation game, the part of B being taken by a man?*

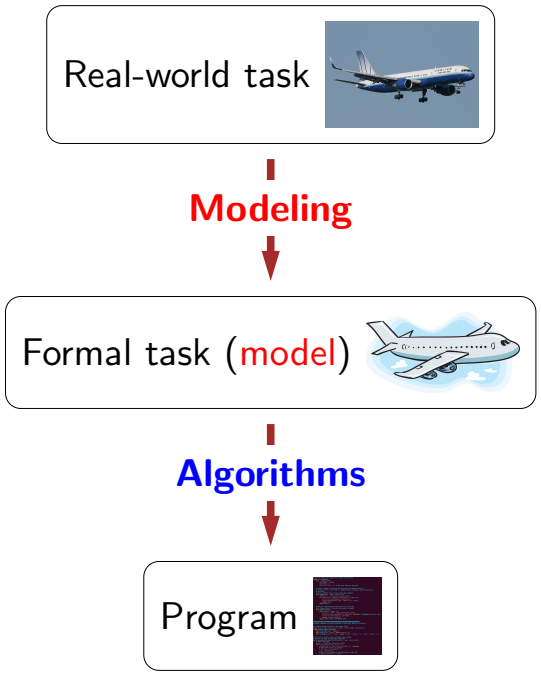
TMs can be quite complex.
But for many, halting is *no* problem — e.g. finite state machines.


Abilities
Goals/Preferences
Prior Knowledge
Stimuli
Past Experiences
Agent
Actions
Environment

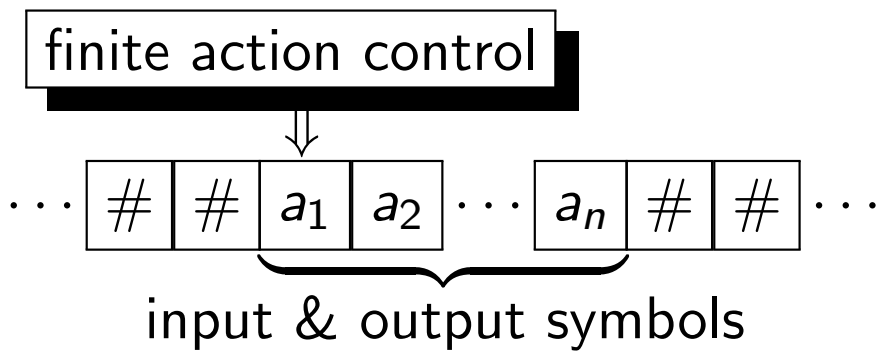

Abilities
Goals/Preferences
Prior Knowledge
Stimuli
Past Experiences
Agent
Actions
Environment


Real-world task
Modeling
Formal task (model)
Algorithms
Program

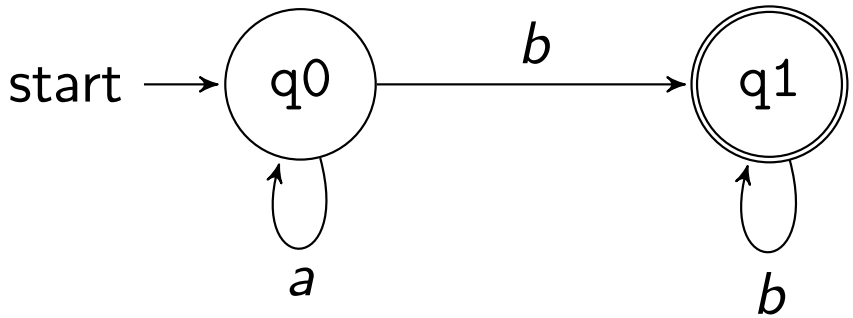


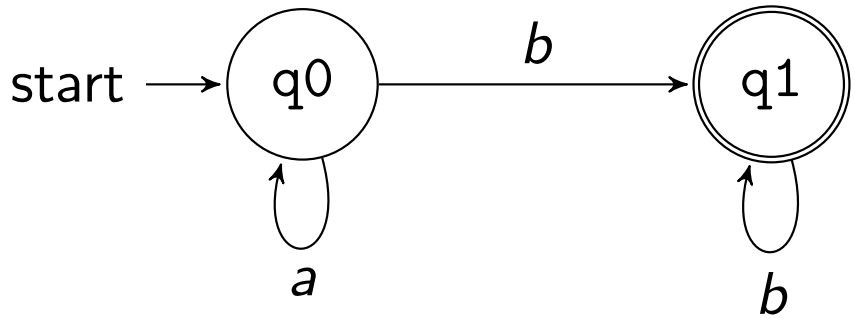

## Church-Turing thesis: Program $\approx$ Turing machine



# Finite state machine (fsm)



# Finite state machine (fsm)




A **fsm** `M` is a triple `[Trans, Final, Q0]` where

- `Trans` is a list of triples `[Q,X,Qn]` such that `M` may, at state `Q` seeing symbol `X`, change state to `Qn`
- `Final` is a list of `M`'s final (i.e. accepting) states
- `Q0` is `M`'s initial state.

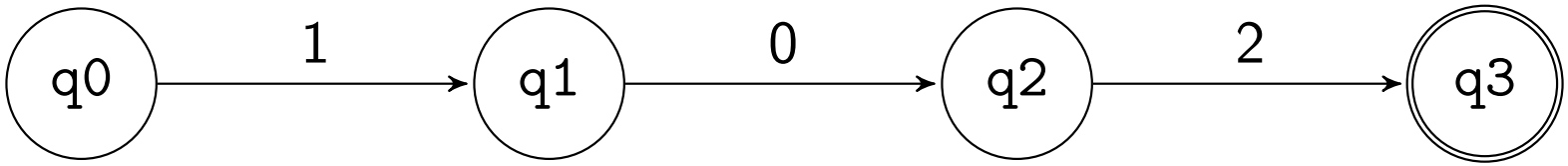
E.g.
```
Trans = [[q0,a,q0],[q0,b,q1],[q1,b,q1]]
Final = [q1]
Q0 = q0
```

# From strings to fsm's

Encode strings as lists; e.g. 102 as `[1,0,2]`.

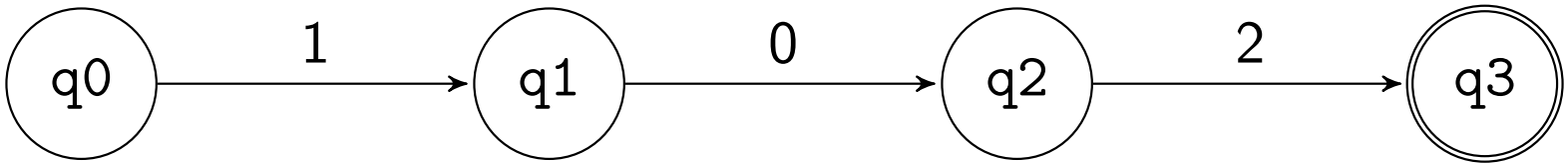
# From strings to fsm's

Encode strings as lists; e.g. 102 as `[1,0,2]`.



# From strings to fsm's

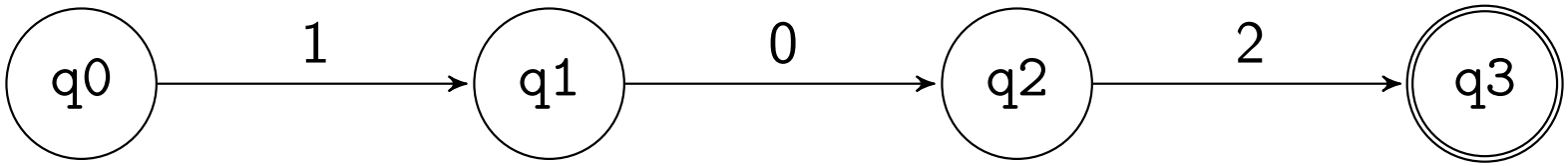
Encode strings as lists; e.g. 102 as `[1,0,2]`.




```
% string2fsm(+String, ?TransitionSet, ?FinalStates)
string2fsm([], [], [q0]).
string2fsm([H|T], Trans, [Last]) :-
```

# From strings to fsm's

Encode strings as lists; e.g. 102 as `[1,0,2]`.




```
% string2fsm(+String, ?TransitionSet, ?FinalStates)
string2fsm([], [], [q0]).
string2fsm([H|T], Trans, [Last]) :-
          mkTL(T, [H], [[q0, H, [H]]], Trans, Last).

% mkTL(+More, +LastSoFar, +TransSoFar, ?Trans, ?Last)
mkTL([], L, Trans, Trans, L).
mkTL([H|T], L, TransSoFar, Trans, Last) :-
     mkTL(T, [H|L], [[L,H,[H|L]]|TransSoFar],
          Trans, Last).
```

States as histories (in reverse)

# More on *states-as-histories*

Encoding `q0` as `[]` leads to the simplification

```
str2fsm(String, Trans, [Last]) :-
            mkTL(String, [], [], Trans, Last).
```

# More on *states-as-histories*

Encoding `q0` as `[]` leads to the simplification

```
str2fsm(String, Trans, [Last]) :-
            mkTL(String, [], [], Trans, Last).
```

*States-as-histories* works for finite languages,

# More on *states-as-histories*

Encoding `q0` as `[]` leads to the simplification

```
str2fsm(String, Trans, [Last]) :-
            mkTL(String, [], [], Trans, Last).
```

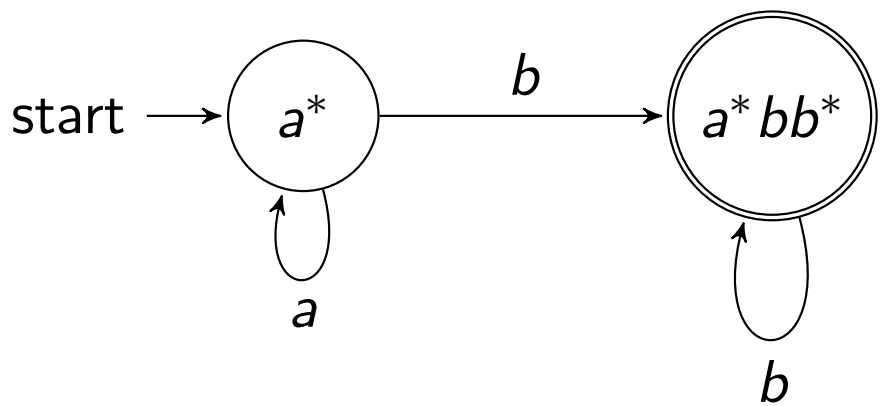
*States-as-histories* works for finite languages, but not say, $a^*bb^*$

# More on *states-as-histories*

Encoding `q0` as `[]` leads to the simplification

```
str2fsm(String, Trans, [Last]) :-
            mkTL(String, [], [], Trans, Last).
```

*States-as-histories* works for finite languages, but not say, $a^*bb^*$




for *state-as-set-of-equivalent-histories*,
where equivalence has to do with acceptable futures ...

# Exercise

Define a 4-ary predicate

```
accept(+Trans,+Final,+Q0,?String)
```

that is true exactly when `[Trans,Final,Q0]` is a fsm that accepts `String` (encoded as a list).

# Exercise

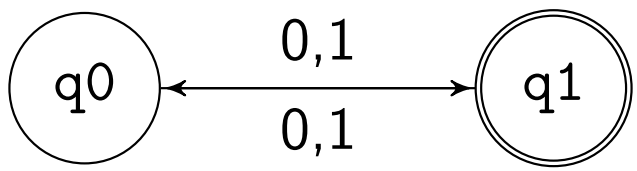
Define a 4-ary predicate

```
accept(+Trans,+Final,+Q0,?String)
```

that is true exactly when `[Trans,Final,Q0]` is a fsm that accepts `String` (encoded as a list).

That is, write a Prolog program to answer queries such as

```
?- accept([[q0,0,q1],[q0,1,q1],[q1,0,q0],[q1,1,q0]],
          [q1], q0, [1,0,0]).
true
```



# Exercise

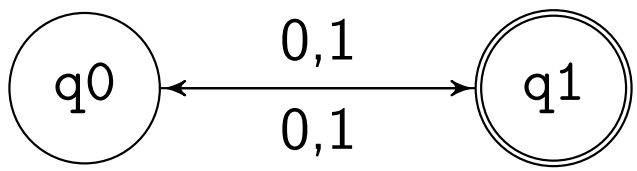
Define a 4-ary predicate

```
accept(+Trans,+Final,+Q0,?String)
```

that is true exactly when `[Trans,Final,Q0]` is a fsm that accepts `String` (encoded as a list).

That is, write a Prolog program to answer queries such as

```
?- accept([[q0,0,q1],[q0,1,q1],[q1,0,q0],[q1,1,q0]],
          [q1], q0, [1,0,0]).
true
```




```
test(String) :- string2fsm(String, Trans, Final),
                accept(Trans, Final, q0, String).
```