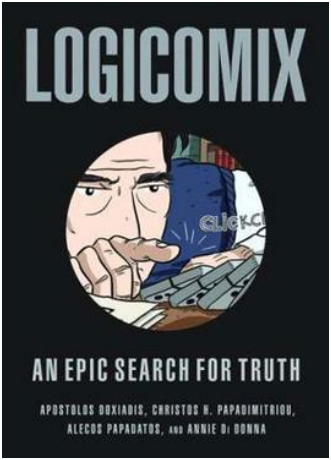
Logic & the search for truth

www.logicomix.com



Logic & the search for truth

Challenges to

- truth

  Liar's Paradox: 'I am lying'

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: cannot count subsets of $\{0, 1, 2, \ldots\}$

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

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

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: cannot count subsets of $\{0, 1, 2, \ldots\}$

- computability
  Turing: Halting Problem

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

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

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

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

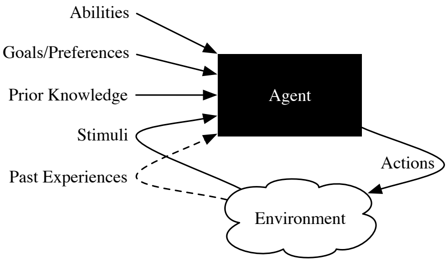
Let

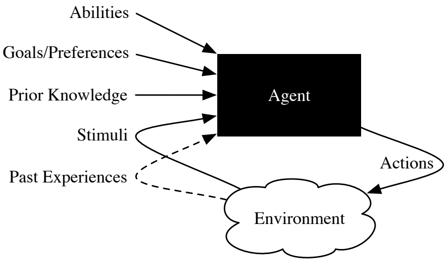
$$\overline{P}(D) \quad :\simeq \quad \begin{cases} 1 & \text{if } P(D, D) = 0 \\ \text{loop} & \text{otherwise.} \end{cases}$$

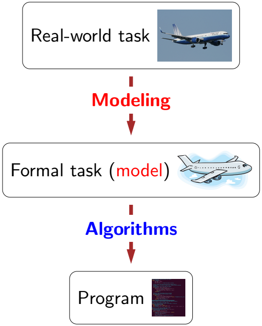
and notice

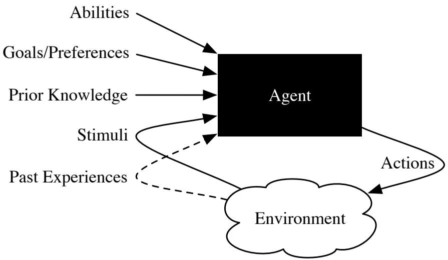
$$\begin{aligned} \mathrm{HP}(\overline{P}, \overline{P}) &= \begin{cases} 1 & \text{if } \overline{P} \text{ halts on } \overline{P} \\ 0 & \text{otherwise} \end{cases} && \text{(def of HP)} \\ &= \begin{cases} 1 & \text{if } P(\overline{P}, \overline{P}) = 0 \\ 0 & \text{otherwise} \end{cases} && \text{(def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

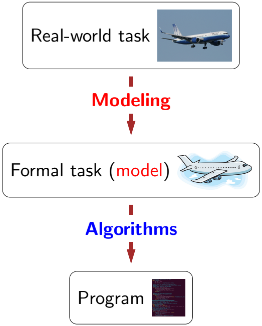
$\square$


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

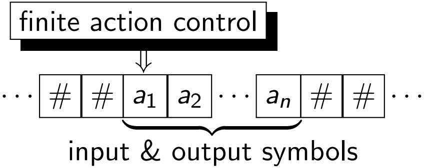

Abilities
Goals/Preferences
Prior Knowledge
Stimuli
Past Experiences
Agent
Actions
Environment


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

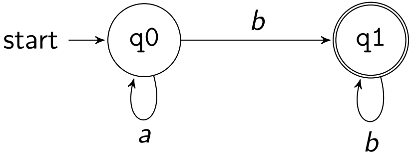




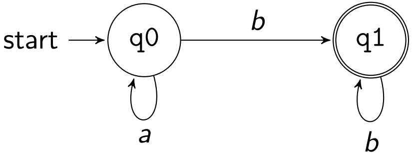
Church-Turing thesis: Program $\approx$ Turing machine



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.

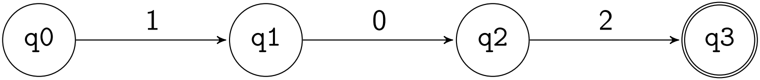
E.g.
```
Trans = [[q0,a,q0],[q0,b,q1],[q1,b,q1]]
Final = [q1]
Q0 = q0
```

From strings to fsm's

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

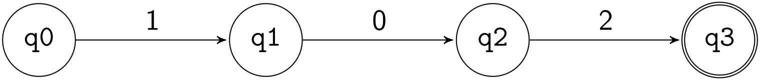
From strings to fsm's

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



From strings to fsm's

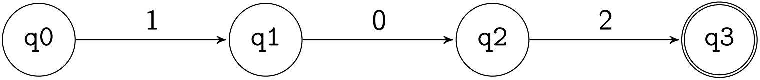
Encode strings as lists; e.g. 102 as `[1,0,2]`.




```
% 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).
```

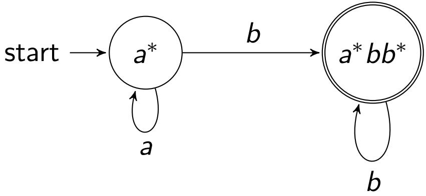
States-as-histories works for finite languages, but not say, $a^* bb^*$

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

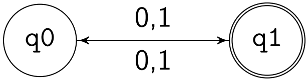
Define a 4-ary predicate

```
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

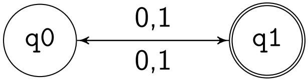
Define a 4-ary predicate

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

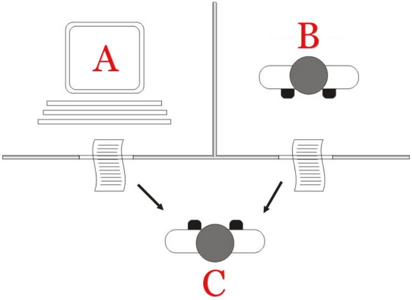
that is true exactly when `[Trans,Final,Q0]` is a fsm that accepts `String` (encoded as a list).

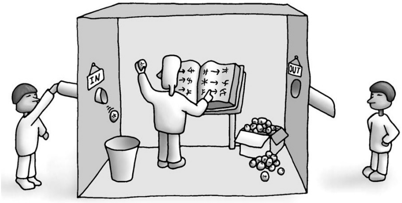
That is, write a Prolog program to answer queries such as

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

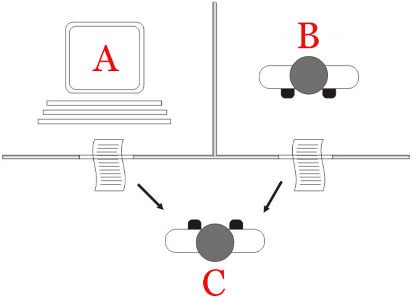


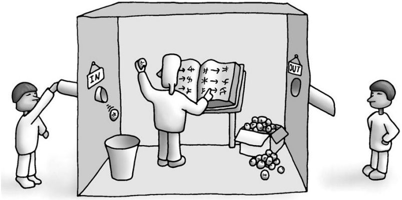

```
test(String) :- string2fsm(String, Trans, Final),
                accept(Trans, Final, q0, String).
```

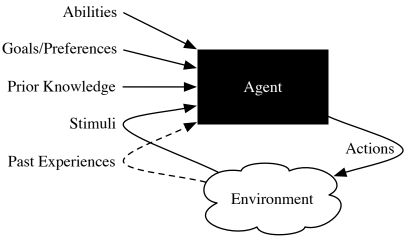

A
B
C


IN
OUT

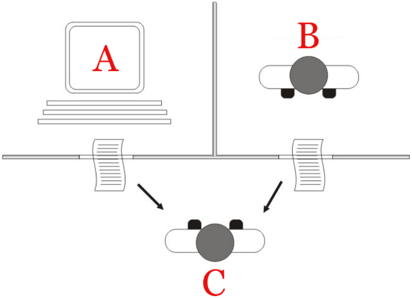
the black box

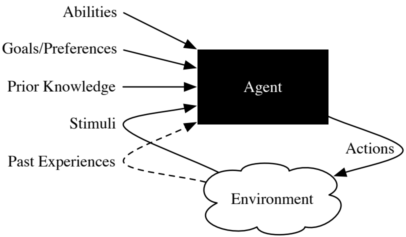


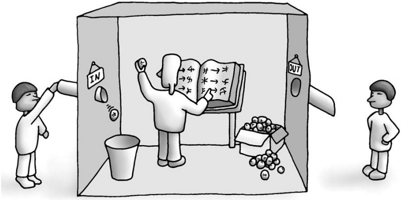




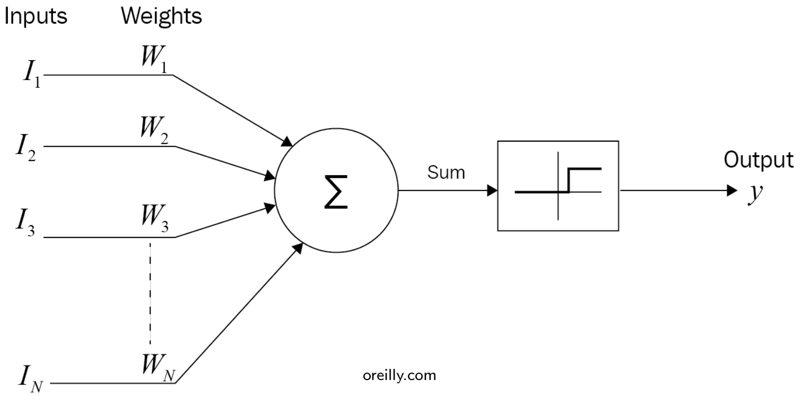
Peering inside the black box







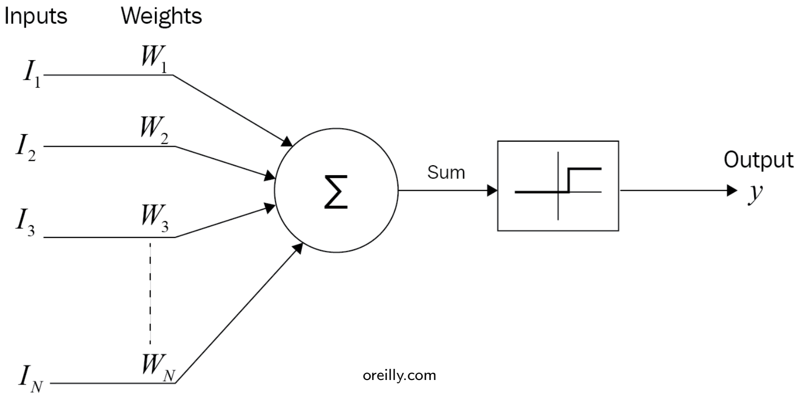
Agents from neurons (perceptrons)



oreilly.com

Agents from neurons (perceptrons)

$$\text{Sum} = W_1 I_1 + W_2 I_2 + \cdots + W_N I_N$$

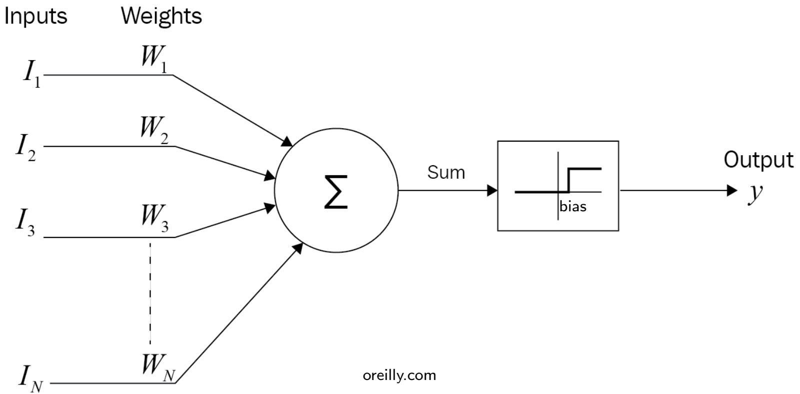


oreilly.com

Agents from neurons (perceptrons)

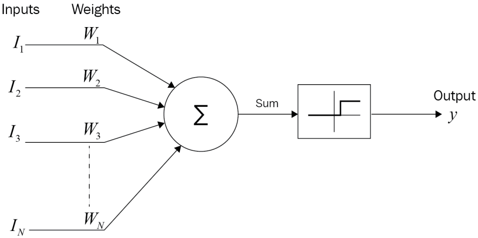
$$\text{Sum} = W_1 I_1 + W_2 I_2 + \cdots + W_N I_N$$

$$y = \begin{cases} 1 & \text{if Sum} \geq \text{bias} \\ 0 & \text{if Sum} < \text{bias} \end{cases}$$

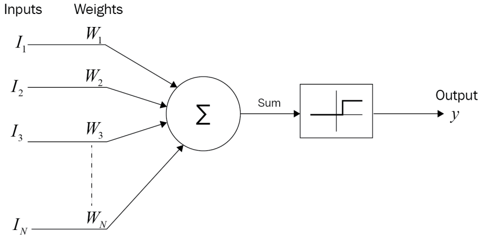


oreilly.com

Finite state aggregations



Finite state aggregations



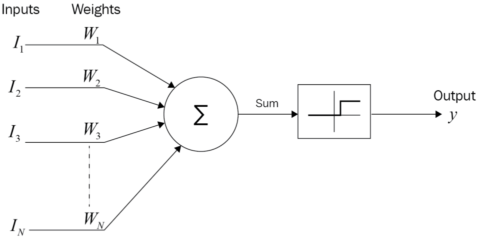
Kleene 1956

$$I_1, I_2 \cdots I_N \;=\; q, a$$

$q \overset{a}{\rightsquigarrow} y \iff$ output y on input a at q

input a given by k *input cells* $\mathrm{N}_1 \ldots \mathrm{N}_k$
state $q = (v_1 \ldots v_m)$ records the values v_i of m *inner cells* $\mathrm{M}_1 \ldots \mathrm{M}_m$

Finite state aggregations



Kleene 1956

$$I_1, I_2 \cdots I_N \;=\; q, a$$

$q \overset{a}{\rightsquigarrow} y \iff$ output y on input a at q

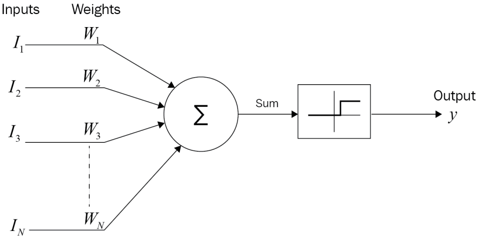
input a given by k *input cells* $\mathrm{N}_1 \ldots \mathrm{N}_k$
state $q = (v_1 \ldots v_m)$ records the values v_i of m *inner cells* $\mathrm{M}_1 \ldots \mathrm{M}_m$

McCulloch & Pitts 1943: neural activity is *all-or-none*

Kleene 1956: input cells are binary, forming a string a of k bits
inner cell M_i can take one of s_i many values

$$q \xrightarrow{a} (y_1 \ldots y_m) \iff q \overset{a}{\rightsquigarrow}_i y_i \text{ for } 1 \leq i \leq m$$

Finite state aggregations



Kleene 1956

$$I_1, I_2 \cdots I_N \;=\; q, a$$

$$q \overset{a}{\rightsquigarrow} y \iff$$

output y on input a at q

input a given by k *input cells* $\mathrm{N}_1 \ldots \mathrm{N}_k$
state $q = (v_1 \ldots v_m)$ records the values v_i of m *inner cells* $\mathrm{M}_1 \ldots \mathrm{M}_m$

McCulloch & Pitts 1943: neural activity is *all-or-none*

Kleene 1956: input cells are binary, forming a string a of k bits
inner cell M_i can take one of s_i many values

$$q \xrightarrow{a} (y_1 \ldots y_m) \iff q \overset{a}{\rightsquigarrow}_i y_i \text{ for } 1 \le i \le m$$

Rabin & Scott 1959: work with any finite sets A, Q and $\rightarrow$

Logical abstractions away from physics and biology

1976 Turing Award to Rabin & Scott for their joint paper

> The internal workings of an automaton will not be analyzed too deeply. We are not concerned with how the machine is built but with what it can do. The definition of the internal structure must be general enough to cover all conceivable machines, but it need not involve itself with problems of circuitry.

Logical abstractions away from physics and biology

1976 Turing Award to Rabin & Scott for their joint paper

> The internal workings of an automaton will not be analyzed too deeply. We are not concerned with how the machine is built but with what it can do. The definition of the internal structure must be general enough to cover all conceivable machines, but it need not involve itself with problems of circuitry. The simple method of obtaining generality without unnecessary detail is to use the concept of internal states. No matter how many wires or tubes or relays the machine contains, its operation is determined by stable states of the machine at discrete time intervals. An actual existing machine may have billions of such internal states, but the number is not important from the theoretical standpoint — only the fact that it is finite.

page 115