MCB 135 Lecture 2
Logic and Turing Completeness

The human brain was once considered unbeatable

The android Data (b. 2338) vs. a novice chess player
Aired in 1992

The tables are turning

IBM's Deep Blue vs. (former) World Chess Champion Garry Kasparov
Rematch of 1997

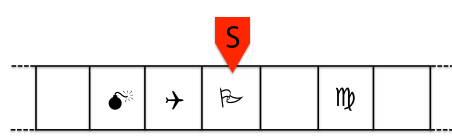
The tables are turning

Today's Outline

- What is Turing completeness?
  - General description of a Turing machine
  - Example: a binary counter
  - Universality and Turing completeness
  - Abilities and limitations of Turing machines
- Proving Turing completeness
  - Boolean functions and logic gates
  - Memory
  - Example from the Game of Life

Turing machine description

- Infinitely-long tape divided into squares
- Each square can contain exactly one symbol from a finite alphabet
- A movable head that can read/write on the tape
- A register with a finite number of states
- A table of instructions specifying what actions to take for all combinations of internal state and current tape symbol



Why is this "design" sensible?

- The tape is a source of input, one option for output, and also a form of *memory*
  – Finite alphabets and states are no limitation since the tape is infinite
- Back-and-forth head movement allows the machine to read/ write memory

Example Turing machine: A (binary) counter

Decimal vs. binary numeration

In decimal, possible digits are 0-9 and place values are powers of ten.
$1101 = 1 \times 10^3 + 1 \times 10^2 + 0 \times 10^1 + 1 \times 10^0 = 1000 + 100 + 1$

In binary, possible digits are 0 or 1 and place values are powers of two.
$1101 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 8 + 4 + 1$

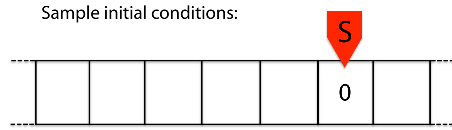
Decimal	Binary
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010

"Counter" Turing Machine

Two internal states:
- "S" for scanning right
- "C" for incrementing the counter

Three tape symbols:
- " " (blank)
- "0"
- "1"

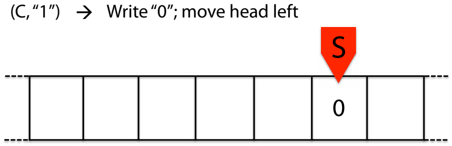
Sample initial conditions:



"Counter" Turing Machine

Update rules:

(S, " ") → Move head left; change state to C
(S, "0") → Move head right
(S, "1") → Move head right
(C, " ") → Write "1"; move head right; change state to S
(C, "0") → Write "1"; move head right; change state to S
(C, "1") → Write "0"; move head left

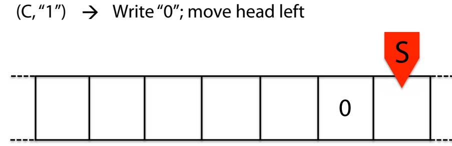


What will the Turing machine do next?

"Counter" Turing Machine

Update rules:

(S, " ") → Move head left; change state to C
(S, "0") → Move head right
(S, "1") → Move head right
(C, " ") → Write "1"; move head right; change state to S
(C, "0") → Write "1"; move head right; change state to S
(C, "1") → Write "0"; move head left

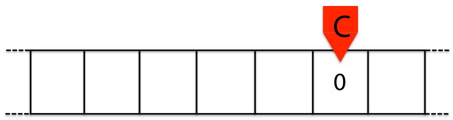


What will the Turing machine do next?

“Counter” Turing Machine

Update rules:

(S, “ ”) → Move head left; change state to C
(S, “0”) → Move head right
(S, “1”) → Move head right
(C, “ ”) → Write “1”; move head right; change state to S
(C, “0”) → Write “1”; move head right; change state to S
(C, “1”) → Write “0”; move head left

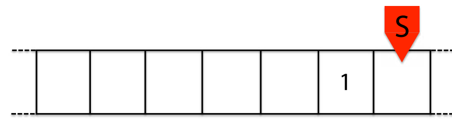


What will the Turing machine do next?

“Counter” Turing Machine

Update rules:

(S, “ ”) → Move head left; change state to C
(S, “0”) → Move head right
(S, “1”) → Move head right
(C, “ ”) → Write “1”; move head right; change state to S
(C, “0”) → Write “1”; move head right; change state to S
(C, “1”) → Write “0”; move head left

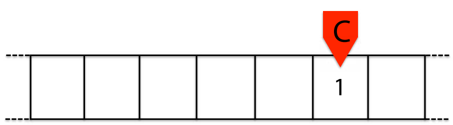


What will the Turing machine do next?

“Counter” Turing Machine

Update rules:

(S, “ ”) → Move head left; change state to C
(S, “0”) → Move head right
(S, “1”) → Move head right
(C, “ ”) → Write “1”; move head right; change state to S
(C, “0”) → Write “1”; move head right; change state to S
(C, “1”) → Write “0”; move head left

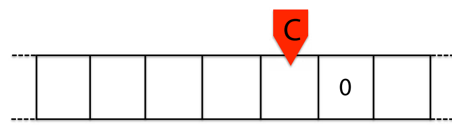


What will the Turing machine do next?

“Counter” Turing Machine

Update rules:

(S, “ ”) → Move head left; change state to C
(S, “0”) → Move head right
(S, “1”) → Move head right
(C, “ ”) → Write “1”; move head right; change state to S
(C, “0”) → Write “1”; move head right; change state to S
(C, “1”) → Write “0”; move head left

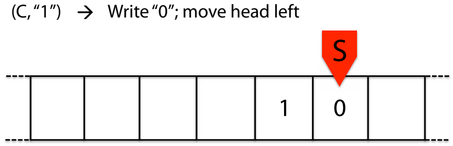


What will the Turing machine do next?

“Counter” Turing Machine

Update rules:

(S, “ ”) → Move head left; change state to C
(S, “0”) → Move head right
(S, “1”) → Move head right
(C, “ ”) → Write “1”; move head right; change state to S
(C, “0”) → Write “1”; move head right; change state to S
(C, “1”) → Write “0”; move head left

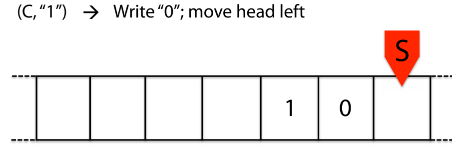


What will the Turing machine do next?

“Counter” Turing Machine

Update rules:

(S, “ ”) → Move head left; change state to C
(S, “0”) → Move head right
(S, “1”) → Move head right
(C, “ ”) → Write “1”; move head right; change state to S
(C, “0”) → Write “1”; move head right; change state to S
(C, “1”) → Write “0”; move head left



What will the Turing machine do next?

“Counter” Turing Machine

Universal Turing Machines

…can behave like any arbitrary Turing machine by first reading its description off the tape.

By definition, any system that can emulate a universal Turing machine can do whatever a Turing machine can. Such systems are called *Turing complete*.

Soon we’ll show that biological systems are Turing complete, but first, let’s learn the limits of Turing machines.

Is there a Turing machine for every problem?

Two sizes (cardinalities) of infinity

- The natural numbers 1, 2, 3, … are *countably infinite*.
- The real numbers are *uncountably infinite*.

What happens when we try to find a one-to-one correspondence between the natural numbers and the real numbers (even just those between 0 and 1, represented in binary)?

Cantor’s diagonality argument: suppose a 1-to-1 map exists

1: 0.0011111010101010101010101000010100…
2: 0.0010101011001001010010110101011 1…
3: 0.1110101001101010101010101000101010…
4: 0.1101010111101010101011101010011110…
…

Can you show that at least one real number between 0 and 1 is missing, for a contradiction?

Cantor’s diagonality argument: suppose a 1-to-1 map exists

1: 0.0011111010101010101010101000010100…
2: 0.00101010110010010100101101010111…
3: 0.1110101001101010101010101000101010…
4: 0.1101010111101010101011101010011110…
…

Define a real number so that its *n*th digit is the *opposite* of that digit in the *n*th real number:

0.1100…

This is a real number, but it appears nowhere in the list! Therefore the 1-to-1 map cannot exist.

Suppose the tape alphabet has *m* symbols and there are *n* internal states.

- What is the number of tape symbol/state combinations (entries in the instruction table)?

$m \times n$

Suppose the tape alphabet has *m* symbols and there are *n* internal states.

- What is the number of tape symbol/state combinations (entries in the instruction table)?

$m \times n$

- How many distinct action combinations can any given instruction have?

$m \times n \times 3 \times 2$

Possible symbols to write

Possible next states

Possible movements

Halt or not

Suppose the tape alphabet has *m* symbols and there are *n* internal states.

- What is the number of tape symbol/state combinations (entries in the instruction table)?

$m \times n$

- How many distinct action combinations can any given instruction have?

$m \times n \times 3 \times 2$

- How many Turing machines of this type are there?

$6m^2n^2$

…the total number is (countably) infinite.

How many problems are there?

- A "problem" has a defined output for all possible inputs
- The number of squares on one tape is countably infinite
- The number of possible inputs is uncountably infinite, like the real numbers.

=> There are more problems than there are Turing machines to solve them.

Some problems that can't be solved with a Turing machine

- Determining whether two arbitrary functions are equivalent
- Determining whether a statement is valid given a set of axioms (Hilbert's *Entscheidungsproblem*)
- Determining whether an arbitrary Turing machine will halt or continue indefinitely given an arbitrary input

---

Still, Turing machines can compute quite a bit.

The Church-Turing conjecture states that Turing machines can compute any function that can be calculated by "mechanical" means (no counterexamples in 75+ years)

A system is Turing complete if it can:

- Implement instruction tables in the form of Boolean functions (to be described shortly)
- Implement arbitrarily-large memory

Sendmail

We can describe all instruction tables using Boolean functions

- The instruction table is a function that maps tape symbol/ current state combinations (inputs) to combinations of actions (outputs).
- We can enumerate all possible inputs as binary numbers, e.g., the inputs for the "counter":

000:	(S, " ")	011:	(C, " ")
001:	(S, "0")	100:	(C, "0")
010:	(S, "1")	101:	(C, "1")

All possible outputs can be represented in binary as well

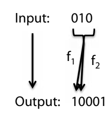
000000	(" ", C, left, halt)	010010	(" ", S, left, halt)
000001	("0", C, left, halt)	010011	("0", S, left, halt)
000010	("1", C, left, halt)	010100	("1", S, left, halt)
000011	(" ", C, stay, halt)	010101	(" ", S, stay, halt)
000100	("0", C, stay, halt)	010110	("0", S, stay, halt)
000101	("1", C, stay, halt)	010111	("1", S, stay, halt)
000110	(" ", C, right, halt)	011000	(" ", S, right, halt)
000111	("0", C, right, halt)	011001	("0", S, right, halt)
001000	("1", C, right, halt)	011010	("1", S, right, halt)
001001	(" ", C, left, continue)	011011	(" ", S, left, continue)
001010	("0", C, left, continue)	011100	("0", S, left, continue)
001011	("1", C, left, continue)	011101	("1", S, left, continue)
001100	(" ", C, stay, continue)	011110	(" ", S, stay, continue)
001101	("0", C, stay, continue)	011111	("0", S, stay, continue)
001110	("1", C, stay, continue)	100000	("1", S, stay, continue)
001111	(" ", C, right, continue)	100001	(" ", S, right, continue)
010000	("0", C, right, continue)	100010	("0", S, right, continue)
010001	(" 1", C, right, continue)	100011	("1", S, right, continue)

The instruction table is a function mapping one binary number to another.

A Boolean function takes a binary number as input and returns either "0" or "1".

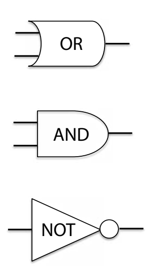
We can define a different Boolean function to specify each digit of the output.

The instruction table is completely described by this set of Boolean functions.



Logic gates implement simple Boolean functions

Variable number of inputs

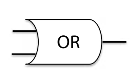


One output (always)

Each individual input/output is:
1 or 0
ON or OFF
TRUE or FALSE

Logic gates implement simple Boolean functions

We can fully describe each gate by its output for each possible input combination



The "OR" gate

Input1	Input2	Output
OFF	OFF	OFF
ON	OFF	ON
OFF	ON	ON
ON	ON	ON

Logic gates implement simple Boolean functions

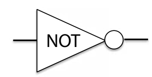
We can fully describe each gate by its output for each possible input combination

The "AND" gate

Input1	Input2	Output
OFF	OFF	OFF
ON	OFF	OFF
OFF	ON	OFF
ON	ON	ON

Logic gates implement simple Boolean functions

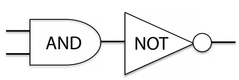
We can fully describe each gate by its output for each possible input combination



The "NOT" gate

Input	Output
OFF	ON
ON	OFF

The last logic gate you'll ever buy



=

The "NAND" gate

Input1	Input2	Output
OFF	OFF	ON
ON	OFF	ON
OFF	ON	ON
ON	ON	OFF

You can implement *any* Boolean function using just NAND gates

The "NAND" gate

Input1	Input2	Output
OFF	OFF	ON
ON	OFF	ON
OFF	ON	ON
ON	ON	OFF

Try it yourselves:

Can you make a two-input OR gate using just NANDs?

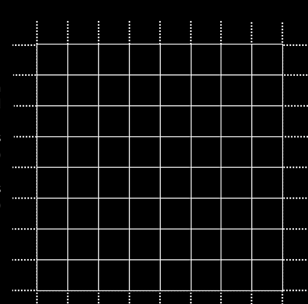
If so, come draw it!

Reminder: the Rules of Life

Played on an infinite 2D grid

Most squares are "dead" (dark)

A few squares are "alive" (light)

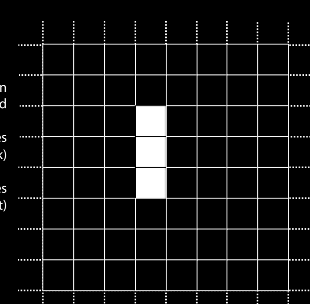


Reminder: the Rules of Life

Played on an infinite 2D grid

Most squares are "dead" (dark)

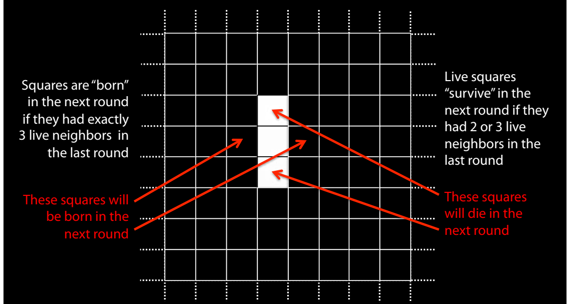
A few squares are "alive" (light)



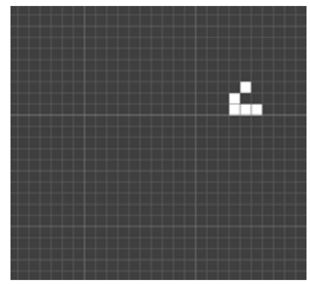
Played in rounds

Rules tell us when a square will die or come to life

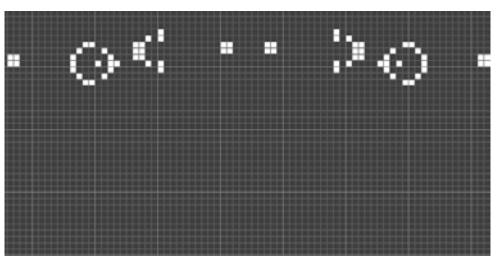
Reminder: the Rules of Life



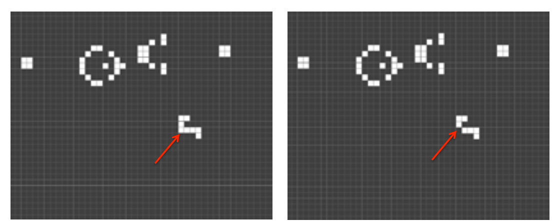
Gliders travel indefinitely: ideal for carrying signals



Gliders (here, fired from glider guns) annihilate on collision



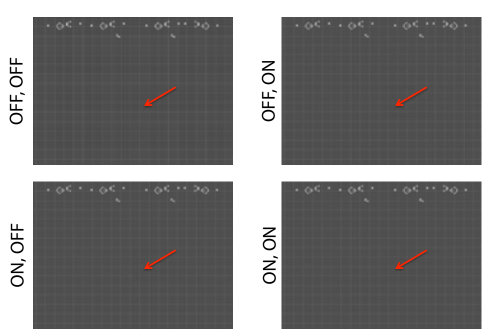
Binary input with the Gosper Glider Gun and stopper



On

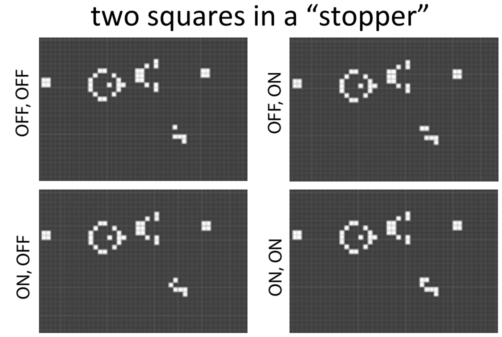
Off

The OR gate implemented in GoL



Credit: Jean-Philippe Rennard

NAND gate implemented using two squares in a "stopper"



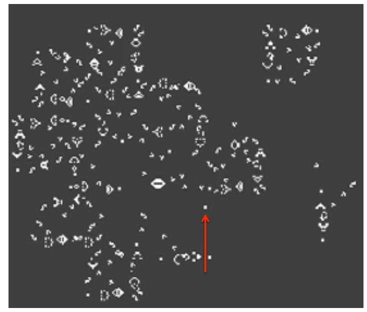
GoL can implement arbitrary Boolean functions, but can it implement memory?

Sliding Block Memory

A *salvo* of gliders can be used to push a *block* one square left or right.

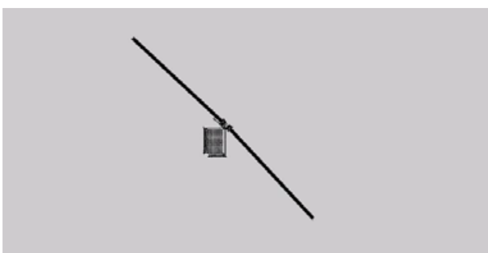
The position of the block can be queried by firing another glider nearby.

If the glider makes it through, then the block must have been in a certain position.



Pattern by Dean Hickerson

Quite literally emulating a universal Turing Machine in the Game of Life



Pattern by Paul Rendell (2010)

Next Time:
Turing completeness in living systems
Self-replication and the role of tape