



A6.9 Turing machines

What this lesson is about

Tape, head, states and transition functions; tracing a Turing machine; the universal Turing machine and why it matters.

- The parts of a Turing machine
- Transition functions
- State transition diagrams
- Hand-tracing
- A Turing machine in Python
- The universal Turing machine

Questions

6 marks in all

1. Which are parts of a Turing machine?

[1 mark]

Tick every answer that is true.

- ☐ A A tape that is infinitely long
- ☐ B A read/write head
- ☐ C A finite set of states including a start state
- ☐ D An unlimited number of states
- ☐ E A transition function

Answer: A, B, C, E. The tape is unlimited but the states and alphabet are finite. That is what separates it from an FSM with a fixed memory.

2. What does $\delta(S2, 0) = (S3, 1, \leftarrow)$ mean?

[1 mark]

- ☐ A In S3 reading 1, write 0, move left and go to S2
- ☐ B In S2 reading 0, write 1, move left and go to S3
- ☐ C In S2 reading 1, write 0, move right and go to S3
- ☐ D In S2 reading 0, write 0, move left and go to S3

Answer: B. The left side is the current state and the symbol read; the right side is the next state, the symbol written and the move.

3. The parity Turing machine adds an even parity bit to a binary string. What is on the tape when it halts, if the input is 10110?

Answer: 101101. 10110 has three 1s, an odd number, so the machine writes 1 at the end.

4. What is a universal Turing machine?

[1 mark]

- ☐ A A Turing machine with an infinite number of states
- ☐ B A Turing machine that can simulate any other Turing machine, given its description and input on the tape
- ☐ C A Turing machine that solves the Halting problem
- ☐ D A real computer built by Turing

Answer: B. Reading another machine's rules as data is the stored program concept, the basis of general-purpose computers.

5. Why are Turing machines important to computer science?

[1 mark]

- ☐ A They are faster than modern computers
- ☐ B They give a definition of what is computable: anything a computer can compute, a Turing machine can too
- ☐ C They can solve the Halting problem
- ☐ D They need no states

Answer: B. A problem that no Turing machine can solve cannot be solved by any computer.

6. What does this program print?

[1 mark]

```
table = {("S0", "1"): ("S0", "0", "R"), ("S0", "0"): ("S0", "1", "R"), ("S0", "_"): ("SH",
    "_", "R")}
tape = dict(enumerate("1100"))
head, state = 0, "S0"
while state != "SH":
    state, write, move = table[(state, tape.get(head, "_"))]
    tape[head] = write
    head = head + 1 if move == "R" else head - 1
print("".join(tape[i] for i in sorted(tape)).strip("_"), head)
```

Answer:

0011 5

The machine flips every bit moving right, then halts on the blank, moving one more cell to position 5.

The task: add one in binary

Write the transition table for a Turing machine that **adds 1** to a binary number, and run it with `run`. - `run(table, text)` is written for you. The head starts on the first (leftmost) symbol in state `"S0"`, the machine halts when it reaches state `"SH"`, blanks are `"_"`, and `run` returns what is left on the tape with the blanks at either end removed. It gives up and returns `None` after 1000 steps. - Write `table`: each key is `(state, symbol)` with `symbol` one of `"0"`, `"1"`, `"_"`, and each value is `(next_state, write, move)` with `move` either `"L"` or `"R"`. Use as many states as you need, with `"S0"` as the start state. - The arithmetic must be done by the machine: no `int`, `bin` or `format`. - For each string in `tests`, in order, print it, `->`, then the result of running your machine on it. For example `1011 -> 1100`. Four lines in all. Think about how you add 1 by hand: where do you start, and what happens to a run of 1s at the end?

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

tests = ["1011", "111", "0", "1001"]

def run(table, text):
    tape = dict(enumerate(text))
    head = 0
    state = "S0"
    for steps in range(1000):
        if state == "SH":
            return "".join(tape[i] for i in sorted(tape)).strip("_")
        symbol = tape.get(head, "_")
        state, write, move = table[(state, symbol)]
        tape[head] = write
        head = head + 1 if move == "R" else head - 1
    return None

table = {}
```

The hint students can ask for: Adding 1 starts at the rightmost digit, but the head starts at the left, so the first state's job is to move right until it finds the blank, then step back. A second state then works leftwards: a 1 becomes 0 and the carry moves on; a 0, or a blank past the left end, becomes 1 and the machine halts.

A solution

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

tests = ["1011", "111", "0", "1001"]

def run(table, text):
    tape = dict(enumerate(text))
    head = 0
    state = "S0"
    for steps in range(1000):
        if state == "SH":
            return "".join(tape[i] for i in sorted(tape)).strip("_")
        symbol = tape.get(head, "_")
        state, write, move = table[(state, symbol)]
        tape[head] = write
        head = head + 1 if move == "R" else head - 1
```

```
        return None

table = {
    ("S0", "0"): ("S0", "0", "R"),
    ("S0", "1"): ("S0", "1", "R"),
    ("S0", "_"): ("S1", "_", "L"),
    ("S1", "1"): ("S1", "0", "L"),
    ("S1", "0"): ("SH", "1", "L"),
    ("S1", "_"): ("SH", "1", "L"),
}
for text in tests:
    print(text, "->", run(table, text))
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.