Automata and Turing Machines

6CCS3COM Computational Models

Josh Murphy

Contents

Formal languages

- An **alphabet** $\mathcal{X}$ is a *finite* set of atomic symbols.
  - $\mathcal{X} = \{x_0, x_1, ..., x_n\}$
  - *e.g.* $B = \{0, 1\}$
  - *e.g.* $C = \{a, b, c, d, ..., z\}$
- A **word** (or string) $w \in \mathcal{X}^*$, over an alphabet $\mathcal{X}$, is a *finite* sequence of symbols taken from $\mathcal{X}$.
  - $\epsilon \in \mathcal{X}^*$, the empty string is a valid word over any alphabet.
  - *e.g.* All binary strings, $B^* = \{0, 1, 00, 01, 10, 11, ...\}$
  - *e.g.* All lowercase character strings, $C^* = \{a, aa, abc, hello, ...\}$
- A **language** $\mathcal{L}$ is a subset of words.
  - $\mathcal{L} \subseteq \mathcal{X}^*$
  - *e.g.* All binary numbers, $N = \{0, 1, 10, 11, ...\} \subseteq B^*$
  - *e.g.* English words, $E = \{a, hello, ...\} \subseteq C^*$
  - Usually defined by a *grammar*, rather than an explicit set.
- **Does a given word belong to a given language?**
  - Solvable by *abstract machines*

Chomsky hierarchy

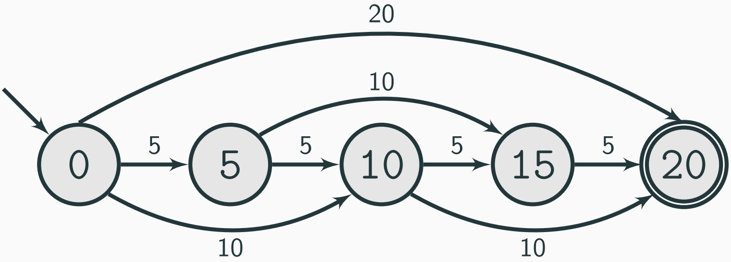
- Depending on the form of the language, identifying valid words can be of varying difficulty.
- The **Chomsky hierarchy** categorises languages into collections according to their difficulty.

	Type	Abstract Machine
	Type-0	Turing Machine
	Type-1	Linear-bounded Turing Machine
	Type-2	Pushdown Automata
Easiest	Type-3	Finite Automata

- Each level of the hierarchy has a corresponding abstract machine, which have varying *memory capabilities*.

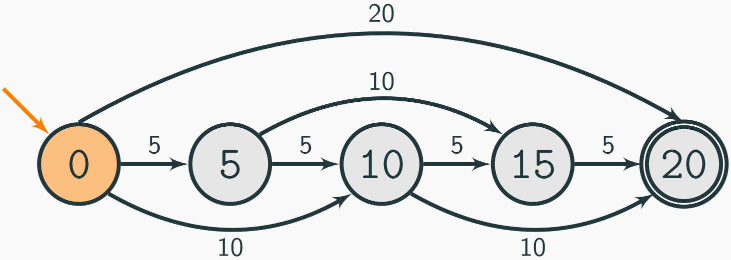
Finite Automata

Example: A vending machine



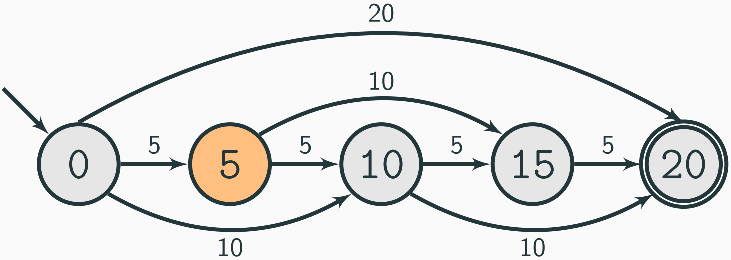
- $\mathcal{X} = \{5p, 10p, 20p\}$
- $\mathcal{L}$ = any sequence of coins adding exactly to 20p.

Example: A vending machine



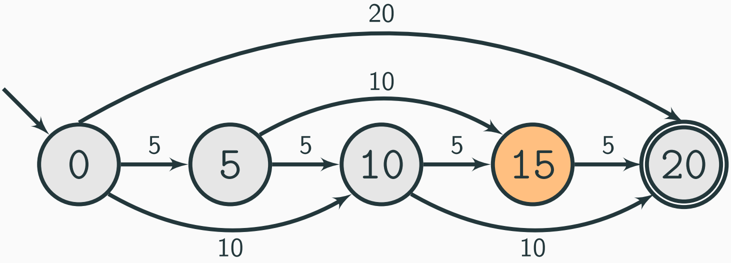
- Is [**5, 10, 5**] a valid word?

Example: A vending machine



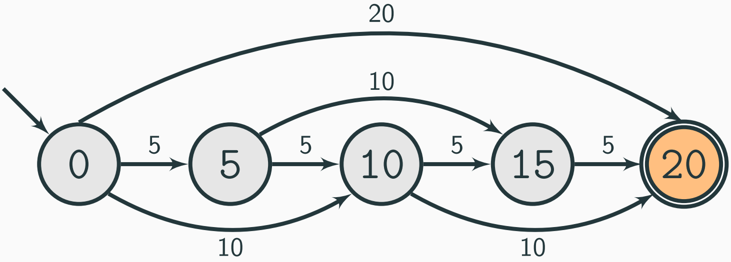
- Is [**5**, **10**, **5**] a valid word?

Example: A vending machine



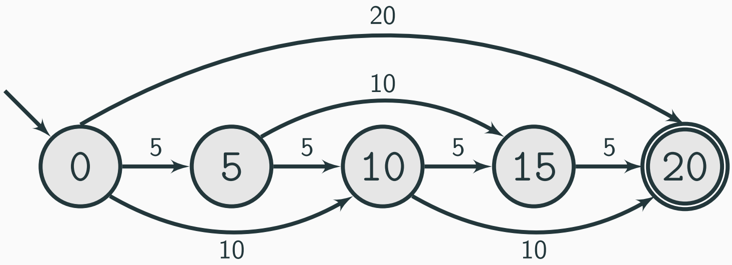
- Is [**5**, **10**, **5**] a valid word?

Example: A vending machine



- Is [**5, 10, 5**] a valid word?
    - Automaton says: **Yes!** It terminates in an accepting state.

Example: A vending machine



- Is [**10, 5**] a valid word?
  - Automaton says: **No!** It terminates in a non-accepting state.
- Is [**10, 5, 10**] a valid word?
  - Automaton says: **No!** There is not a valid transition.

Finite automata: formal definition

- A **finite automaton** is a tuple $(\mathcal{X}, \mathcal{Q}, q_i, F, \delta)$, where:
  - $\mathcal{X}$ is an alphabet (finite set of atomic symbols);
  - $\mathcal{Q} = \{q_0, q_1, ..., q_n\}$ is a finite set of states;
  - $q_i \in \mathcal{Q}$ is the initial state;
  - $F \subseteq \mathcal{Q}$ is the set of accepting states; and
  - $\delta : \mathcal{Q} \times \mathcal{X} \to \mathcal{Q}$ is the transition function.
- The language **associated** with a finite automaton A is the set of words it accepts, denoted $L(A)$.

Exercise

1. Draw the automaton $(\{a\}, \{q_0, q_1\}, q_0, \{q_0\}, \delta)$ where:
    - $\delta(q_0, a) = q_1$
    - $\delta(q_1, a) = q_0$

2. Give an informal description of the language the above automaton is associated with.

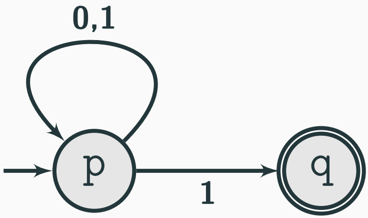
3. Build a finite automaton, with $\mathcal{X} = \{0, 1\}$, to recognise:
    - The language of the strings of 0s of any length.
    - The language of the strings of 0s and 1s that contain a 1 in the 2nd position.

Non-deterministic finite automata

- Our previous definition of finite automata was **deterministic**: the co-domain of the transition was a single state.
- Finite automata can also be **non-deterministic**: the co-doamin of the transition can be a set of states.

- A **non-deterministic finite automaton** is a tuple $(\mathcal{X}, \mathcal{Q}, q_i, F, \delta)$, where:
  - $\mathcal{X}$ is an alphabet (finite set of atomic symbols);
  - $\mathcal{Q} = \{q_0, q_1, ..., q_n\}$ is a finite set of states;
  - $q_i \in \mathcal{Q}$ is the initial state;
  - $F \subseteq \mathcal{Q}$ is the set of accepting states; and
  - $\delta : \mathcal{Q} \times (\mathcal{X} \cup \epsilon) \rightarrow \mathcal{P}(\mathcal{Q})$ is the transition function, where:
    - $\mathcal{P}(\mathcal{Q})$ is the power set of $\mathcal{Q}$).
    - ϵ is an empty transition label that can be taken at any point when reading a word.
- A word is accepted if there is **any** trace from the initial state to an accepting state.

Example: Non-deterministic finite automata



- Is **1011** a valid word?
  - $p \rightarrow^1 q$, non-accepting trace
  - $p \rightarrow^1 p \rightarrow^0 p \rightarrow^1 q$, non-accepting trace
  - $p \rightarrow^1 p \rightarrow^0 p \rightarrow^1 p \rightarrow^1 q$, **accepting trace**
  - $p \rightarrow^1 p \rightarrow^0 p \rightarrow^1 p \rightarrow^1 p$, non-accepting trace
- Automaton says: **Yes!** There is at least one accepting trace.

Non-deterministic vs. deterministic finite automata

- Non-deterministic and deterministic finite automata are **computationally equivalent**.
    - A language can be recognised by a non-deterministic finite automata iff the language can be recognised by a deterministic finite automata.

- **Exercise:** Build a deterministic finite automaton that recognises the same language as the non-deterministic automaton on the previous slide.

The power of finite automata

- Finite automata have no ‘memory’.
  - They can have no knowledge of previous state.
- This limits the languages a finite automata can be associated with.
  - Only the simplest languages in the Chomsky hierarchy can be associated with them.
  - These are called **regular languages**.
- **So, what determines if a language is regular?**
  - Regular languages are **closed under basic set operations** (*e.g.* union and intersection).
  - A language that is not **pumpable** is not regular. In other words, a regular language must be pumpable.

Pumping Lemma

Not examinable

Pumping Lemma:

- Let L be a regular language. There exists a constant n such that if z is any given word in L with more than n symbols, then there are three words u, v, w such that z can be written as the concatenation uvw where:
  - the length of uv is less than or equal to n,
  - the length of v is greater than or equal to 1, and
  - for any $i \geq 0$, $uv^iw \in L$ where v^i represents the word v repeated i times.

- If a language does not obey the pumping lemma then it cannot be associated with a finite automata.

Exercise

1. Build a finite automata that recognises the language, with $\mathcal{X} = \{(,)\}$, that has balanced parentheses.
    - *e.g.* (()()) is recognised but ((())() is not.

Pushdown Automata

Pushdown automata

- **Pushdown automata (PDA)** are finite automata but with a **stack**.
    - This makes them more powerful, and allows them to be associated with more languages in the Chomsky hierarchy.

Stacks

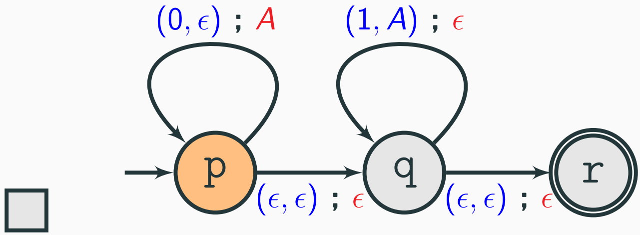
- Stacks are a specialised form of memory, where:
  - only the top element of the stack can be read (**pop**) and in the process it is removed from the stack, and
  - new elements must be added to the top of the stack (**push**).

- Stacks in PDA store **symbols**.
  - The stack starts empty.
  - Each transition, a symbol can be popped from the stack, and a symbol can be pushed to the stack.
  - We assume we can always push and pop empty ϵ on the stack, without modifying the stack.

PDA: formal definition

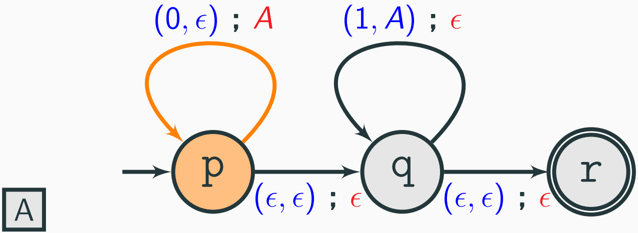
- A **PDA** is a tuple $(\mathcal{X}, \mathcal{Q}, \Gamma, q_i, F, \delta)$, where:
  - $\mathcal{X}$ is an alphabet (finite set of atomic symbols);
  - $\mathcal{Q} = \{q_0, q_1, ..., q_n\}$ is a finite set of states;
  - Γ is the set of symbols that can be stored on the stack;
  - $q_i \in \mathcal{Q}$ is the initial state;
  - $F \subseteq \mathcal{Q}$ is the set of accepting states; and
  - $\delta : \mathcal{Q} \times (\mathcal{X} \cup \epsilon) \times \underbrace{(\Gamma \cup \epsilon)}_{\textbf{Pop}} \to \mathcal{Q} \times \underbrace{(\Gamma \cup \epsilon)}_{\textbf{Push}}$ is the transition function.
- The transition function now maps a (state, letter, symbol) to a (state, symbol).
  - The symbol in the domain is the symbol to be popped.
  - The symbol in the codomain is the symbol to be pushed.
  - If the symbol is ϵ (empty) then nothing is to be popped/pushed.
- We also allow transitions to be labelled ϵ.
  - Note: this makes PDAs non-deterministic.
- A word is accepted if there is **any** trace from the initial state to an accepting state.

Example: PDA



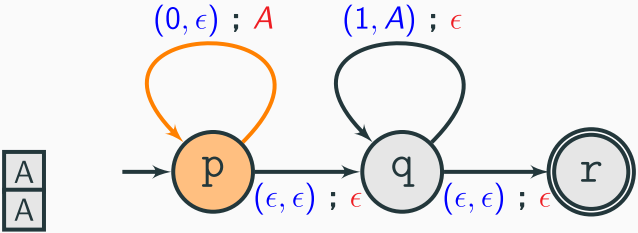
- Is **0011** a valid word?
- Transition notation: (a, b) ; c
  - a : the next character of the word
  - b : the symbol to be popped from the stack
  - c : the symbol to be pushed to the stack

Example: PDA



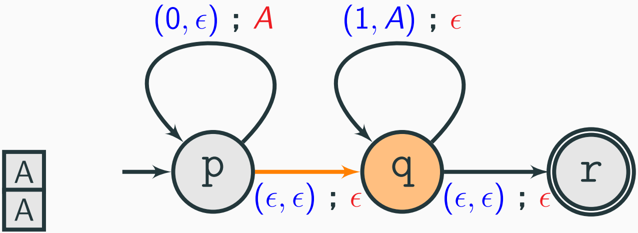
- Is **0011** a valid word?

Example: PDA



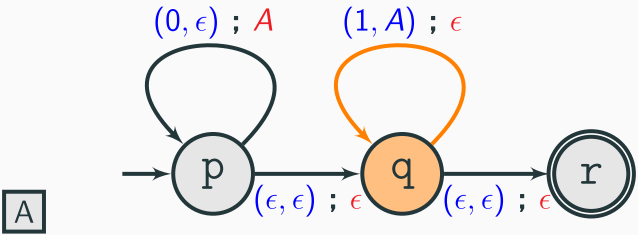
- Is **0011** a valid word?

Example: PDA



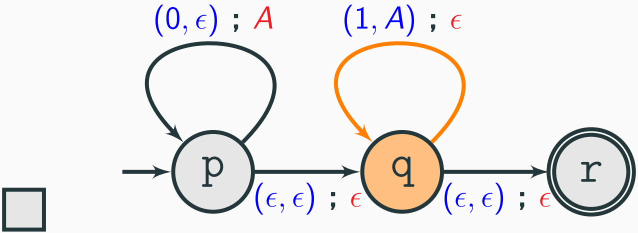
- Is $\mathbf{00}\epsilon\mathbf{11}$ a valid word?

Example: PDA



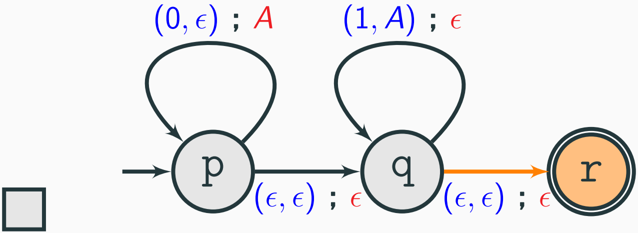
- Is **0011** a valid word?

Example: PDA



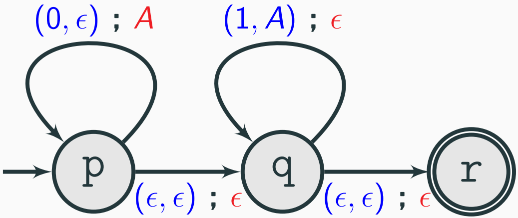
- Is **0011** a valid word?

Example: PDA



- Is $\mathbf{0011}\epsilon$ a valid word?
  - Automaton says: **Yes!**

Exercises



1. Is ϵ a valid word?
2. Is **1100** a valid word?
3. Is **00011** a valid word?
4. **Challenge:** Build a PDA that recognises the language, with $\mathcal{X} = \{(,)\}$, that has balanced parentheses.

The power of PDA

- PDA have a limited (stack) memory.
- This limits the languages a PDA can be associated with.
  - PDA are strictly more powerful than finite automata.
  - PDA can recognise Type-2 and Type-3 languages in the Chomsky hierarchy.
  - These are called **context-free languages**.
- **So, what determines if a language is context-free?**
  - There is a similar pumping lemma for PDA that demonstrate their limitation (see textbook, p.23, for intuition of this proof).

Turing Machines

Turing Machines

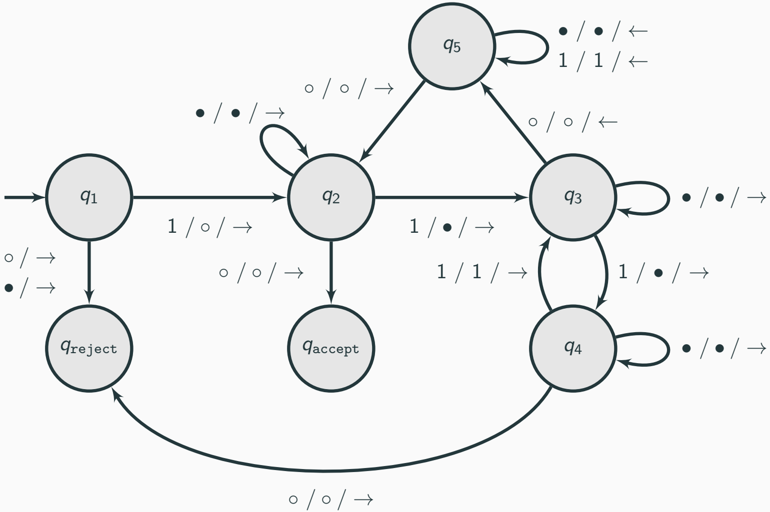
- **Turing Machines (TM)** are similar to finite automata, but they have an **infinite tape** for memory.
- TMs also have a **head** that moves left and right along the tape.
    - The head can **write** a symbol to the tape.
    - The head can **read** a symbol from the tape.

TM: formal definition

- A **TM** is a tuple $(\mathcal{X}, \mathcal{Q}, \Gamma, q_i, F, \delta)$, where:
  - $\mathcal{X}$ is an alphabet (finite set of atomic symbols);
  - $\mathcal{Q} = \{q_0, q_1, ..., q_n\}$ is a finite set of states;
  - Γ is the set of symbols that can be stored on the tape, we assume $\{\circ, \bullet\} \subseteq \Gamma$, which are *blank* and *marker* symbols;
  - $q_i \in \mathcal{Q}$ is the initial state;
  - $\{q_{\text{accept}}, q_{\text{reject}}\} \subseteq \mathcal{Q}$ are terminating states; and
  - $\delta : \mathcal{Q} \times \Gamma \to \mathcal{Q} \times \Gamma \times \{\leftarrow, \rightarrow\}$ is the transition function.

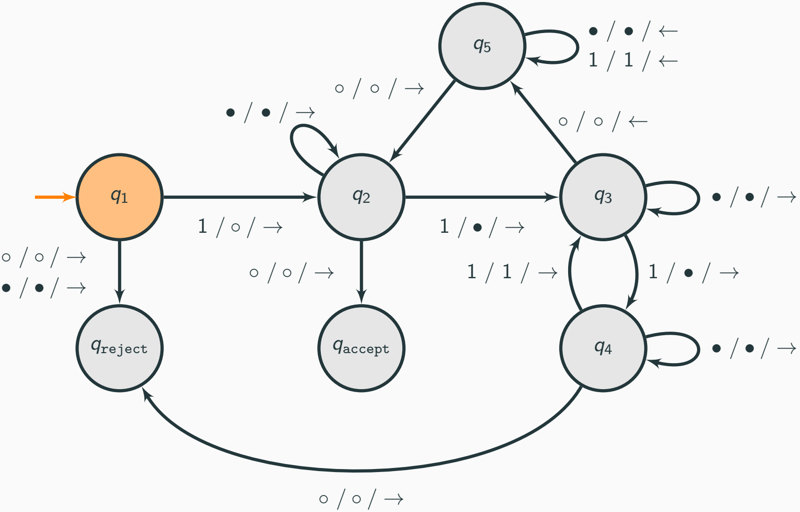
- The transition function now maps a (state, symbol) to a (state, symbol, direction).
  - The direction tells the head which way to move.
  - Transition notation: read / write / move
- The initial state of the tape is the input word.
- Iff the TM reaches q_{accept} then the word is accepted.

TM Example



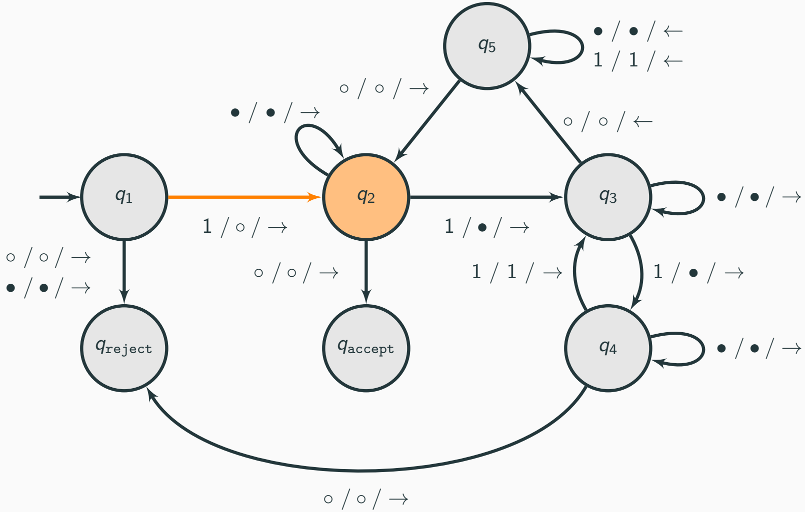
q_5
$\bullet$ / $\bullet$ / $\leftarrow$
1 / 1 / $\leftarrow$
$\circ$ / $\circ$ / $\rightarrow$
$\bullet$ / $\bullet$ / $\rightarrow$
$\circ$ / $\circ$ / $\leftarrow$
q_1
q_2
q_3
$\bullet$ / $\bullet$ / $\rightarrow$
1 / $\circ$ / $\rightarrow$
1 / $\bullet$ / $\rightarrow$
$\circ$ / $\circ$ / $\rightarrow$
$\bullet$ / $\bullet$ / $\rightarrow$
$\circ$ / $\circ$ / $\rightarrow$
1 / 1 / $\rightarrow$
1 / $\bullet$ / $\rightarrow$
$q_{\texttt{reject}}$
$q_{\texttt{accept}}$
q_4
$\bullet$ / $\bullet$ / $\rightarrow$
$\circ$ / $\circ$ / $\rightarrow$

TM Example



⋯	$\circ$	1	1	$\circ$	⋯

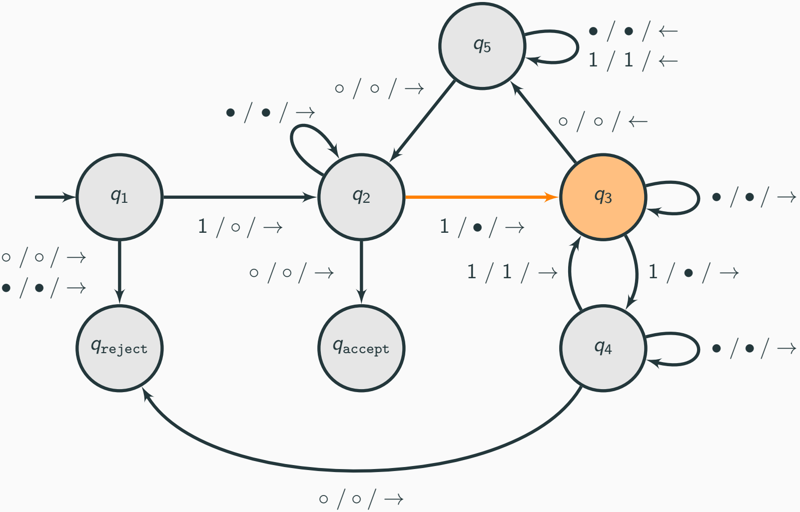
TM Example



q_5
$\bullet$ / $\bullet$ / $\leftarrow$
1 / 1 / $\leftarrow$
$\circ$ / $\circ$ / $\rightarrow$
$\bullet$ / $\bullet$ / $\rightarrow$
$\circ$ / $\circ$ / $\leftarrow$
q_1
q_2
q_3
$\bullet$ / $\bullet$ / $\rightarrow$
1 / $\circ$ / $\rightarrow$
1 / $\bullet$ / $\rightarrow$
$\circ$ / $\circ$ / $\rightarrow$
$\bullet$ / $\bullet$ / $\rightarrow$
$\circ$ / $\circ$ / $\rightarrow$
1 / 1 / $\rightarrow$
1 / $\bullet$ / $\rightarrow$
$q_{\texttt{reject}}$
$q_{\texttt{accept}}$
q_4
$\bullet$ / $\bullet$ / $\rightarrow$
$\circ$ / $\circ$ / $\rightarrow$

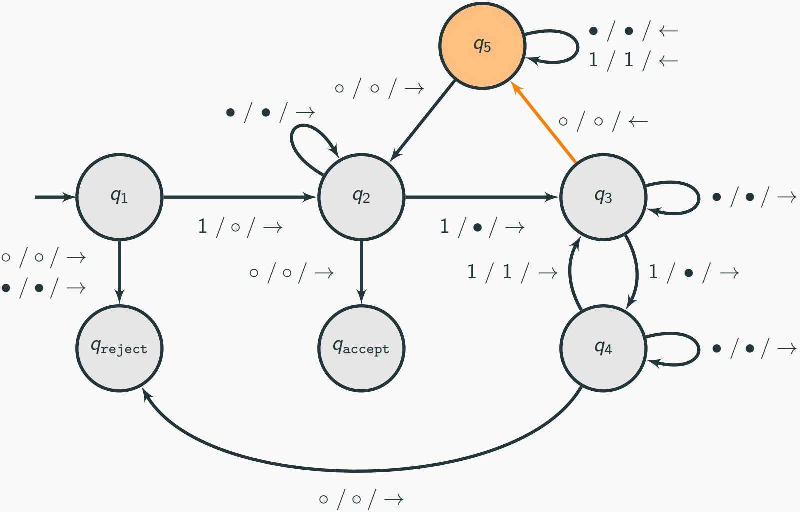
$\cdots$ $\circ$ $\circ$ 1 $\circ$ $\cdots$

TM Example

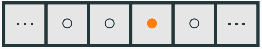


⋯	$\circ$	$\circ$	$\bullet$	$\circ$	⋯

TM Example

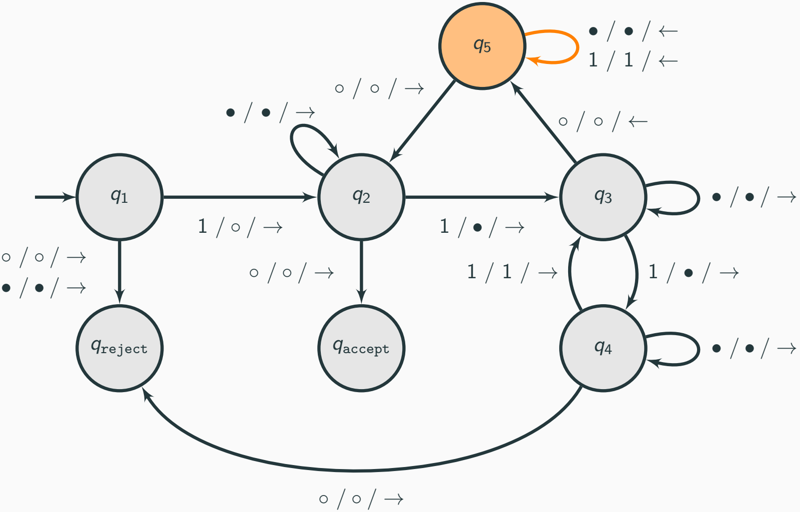


q_5
$\bullet$ / $\bullet$ / $\leftarrow$
1 / 1 / $\leftarrow$
$\circ$ / $\circ$ / $\rightarrow$
$\bullet$ / $\bullet$ / $\rightarrow$
$\circ$ / $\circ$ / $\leftarrow$
q_1
q_2
q_3
$\bullet$ / $\bullet$ / $\rightarrow$
1 / $\circ$ / $\rightarrow$
1 / $\bullet$ / $\rightarrow$
$\circ$ / $\circ$ / $\rightarrow$
$\bullet$ / $\bullet$ / $\rightarrow$
$\circ$ / $\circ$ / $\rightarrow$
1 / 1 / $\rightarrow$
1 / $\bullet$ / $\rightarrow$
$q_{\texttt{reject}}$
$q_{\texttt{accept}}$
q_4
$\bullet$ / $\bullet$ / $\rightarrow$
$\circ$ / $\circ$ / $\rightarrow$



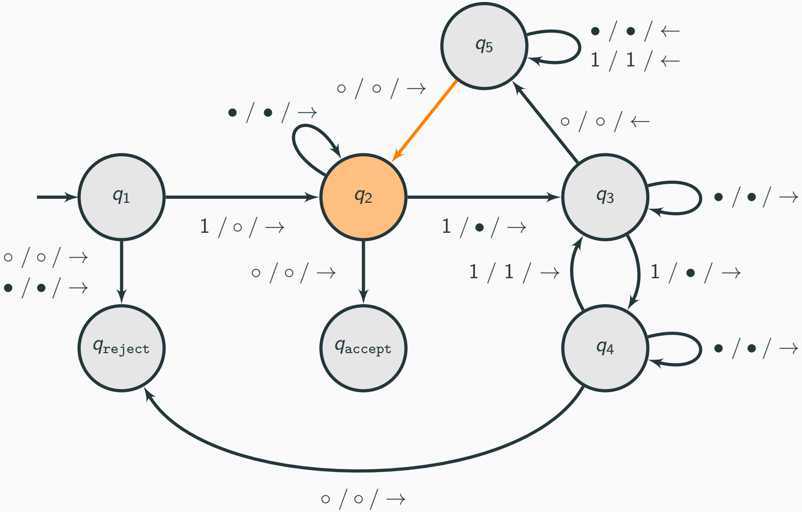
$\cdots$ $\circ$ $\circ$ $\bullet$ $\circ$ $\cdots$

TM Example

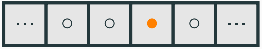


$\cdots$	$\circ$	$\circ$	$\bullet$	$\circ$	$\cdots$

TM Example

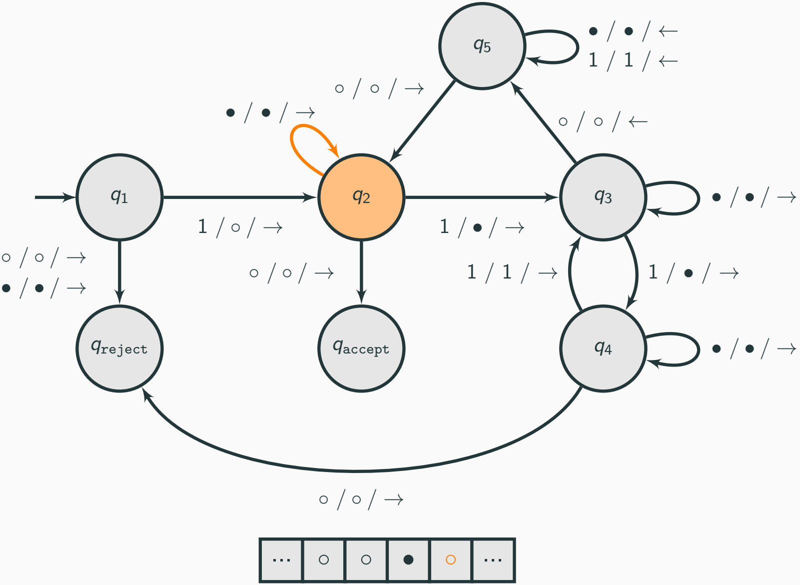


q_5
$\bullet$ / $\bullet$ / $\leftarrow$
1 / 1 / $\leftarrow$
$\circ$ / $\circ$ / $\rightarrow$
$\bullet$ / $\bullet$ / $\rightarrow$
$\circ$ / $\circ$ / $\leftarrow$
q_1
q_2
q_3
$\bullet$ / $\bullet$ / $\rightarrow$
1 / $\circ$ / $\rightarrow$
1 / $\bullet$ / $\rightarrow$
$\circ$ / $\circ$ / $\rightarrow$
$\bullet$ / $\bullet$ / $\rightarrow$
$\circ$ / $\circ$ / $\rightarrow$
1 / 1 / $\rightarrow$
1 / $\bullet$ / $\rightarrow$
$q_{\mathtt{reject}}$
$q_{\mathtt{accept}}$
q_4
$\bullet$ / $\bullet$ / $\rightarrow$
$\circ$ / $\circ$ / $\rightarrow$



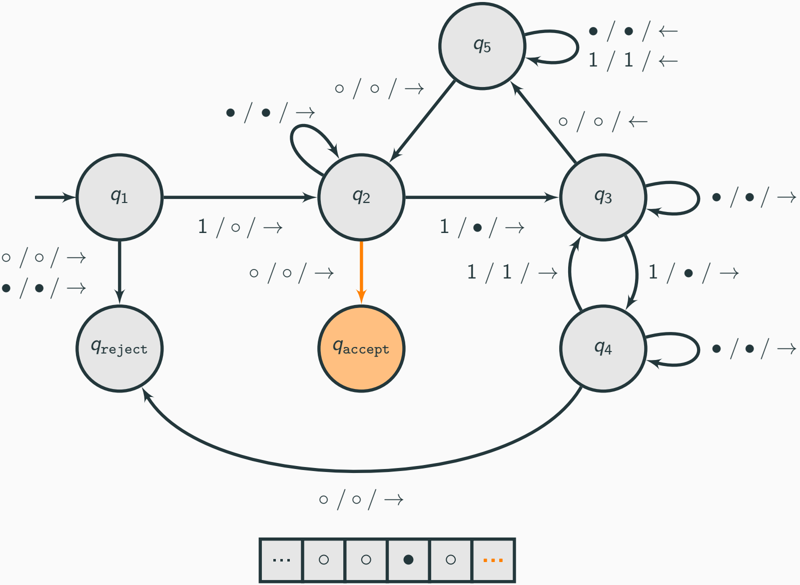
… $\circ$ $\circ$ $\bullet$ $\circ$ …

TM Example

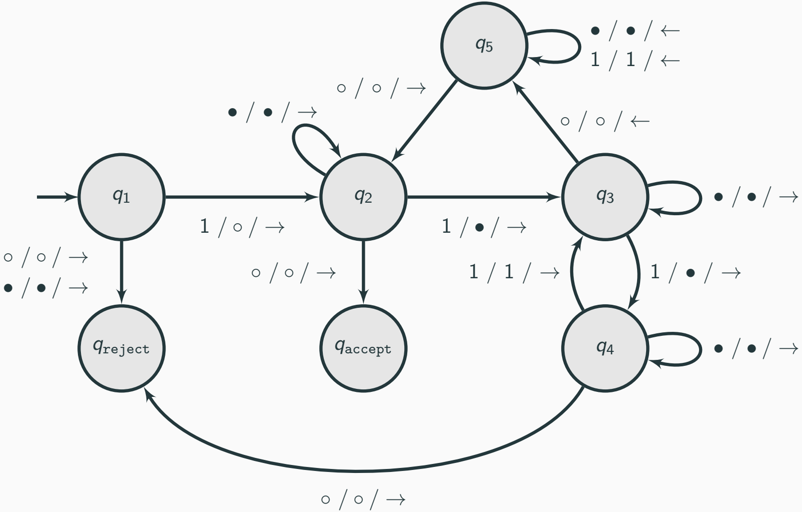


q_5
$\bullet$ / $\bullet$ / $\leftarrow$
1 / 1 / $\leftarrow$
$\circ$ / $\circ$ / $\rightarrow$
$\bullet$ / $\bullet$ / $\rightarrow$
$\circ$ / $\circ$ / $\leftarrow$
q_1
q_2
q_3
$\bullet$ / $\bullet$ / $\rightarrow$
1 / $\circ$ / $\rightarrow$
1 / $\bullet$ / $\rightarrow$
$\circ$ / $\circ$ / $\rightarrow$
$\bullet$ / $\bullet$ / $\rightarrow$
$\circ$ / $\circ$ / $\rightarrow$
1 / 1 / $\rightarrow$
1 / $\bullet$ / $\rightarrow$
$q_{\texttt{reject}}$
$q_{\texttt{accept}}$
q_4
$\bullet$ / $\bullet$ / $\rightarrow$
$\circ$ / $\circ$ / $\rightarrow$
$\cdots$ $\circ$ $\circ$ $\bullet$ $\circ$ $\cdots$

TM Example


q_5
$\bullet$ / $\bullet$ / $\leftarrow$
1 / 1 / $\leftarrow$
$\circ$ / $\circ$ / $\rightarrow$
$\bullet$ / $\bullet$ / $\rightarrow$
$\circ$ / $\circ$ / $\leftarrow$
q_1
q_2
q_3
$\bullet$ / $\bullet$ / $\rightarrow$
1 / $\circ$ / $\rightarrow$
1 / $\bullet$ / $\rightarrow$
$\circ$ / $\circ$ / $\rightarrow$
$\bullet$ / $\bullet$ / $\rightarrow$
$\circ$ / $\circ$ / $\rightarrow$
1 / 1 / $\rightarrow$
1 / $\bullet$ / $\rightarrow$
$q_{\texttt{reject}}$
$q_{\texttt{accept}}$
q_4
$\bullet$ / $\bullet$ / $\rightarrow$
$\circ$ / $\circ$ / $\rightarrow$
… $\circ$ $\circ$ $\bullet$ $\circ$ …

TM Example



- **Exercise:** What is the language associated with this TM?

Variants of TM

- There are many variations to TMs, for example:
    - Non-deterministic transitions.
    - Different head movements.
    - Multiple tapes.
    - Multi-dimensional tapes.
    - Symbols (purest form has just two symbols: 0, 1).
- All variants of TMs are **computationally equivalent**.

The Universal TM

- Turing machines can be encoded onto a tape.
- This allows a TM U to take another TM A as input.
  - U can simulate A
  - U can attempt to establish properties of A (*e.g.* if it halts or not.
  - U is called **Universal Turing Machine**.

The power of TM

- TMs are strictly more powerful than PDA.
- TMs can be associated with Type-0 languages in the Chomsky hierarchy.
- These are called **recursively-enumerable languages**.

- However, remember the **Halting Problem**!
    - There are languages that are not recognisable, even by a TM!

TMs as partial functions

- TMs implement **partial functions**.
  - They map words to (word, boolean) tuples.
- Since TMs can get caught in continuous loops, its possible for a TM to not return an output for a given word.
  - Hence why they are only partial functions.
- A total function that can be implemented by a TM is called **Turing computable**.
- Turing and Church proved all computable functions can be defined in terms of TM.
  - If a function is computable, it is Turing computable!

TMs and Imperative Programming

- TMs are the theoretical basis of **imperative programming**.
    - *e.g.* Java, Python, C...
- Soon, we will look at an equivalent model of computation that is the basis of **functional programming**
    - *e.g.* Haskel, Lisp, WolframAlpha...