Models of Computation

Tom Melham

Oxford University Computing Laboratory

Based on material developed by Luke Ong and Hanno Nickau

Introduction: Automata, Computability and Complexity

Key question: *What are the capabilities and limitations of computers?*

We seek mathematically precise answers.

Complexity Theory. Easy problem: sorting. Hard problem: scheduling.

What makes some problems computationally hard and others easy?

Computability Theory. *Which problems are solvable by computers and which are not?*

Both Complexity Theory and Computability Theory require a precise definition of a *computer*.

Automata Theory deals with definitions and properties of mathematical models of *computation*.

Finite Automata

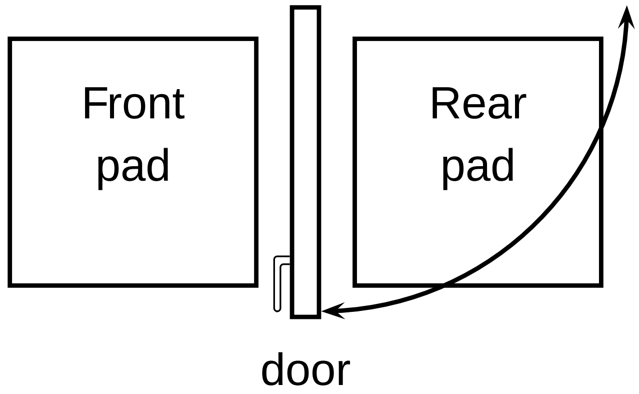
Theory of Computation begins with “What is a model of computation?”

A computational model may be accurate in some ways, but not in others.

We begin with the simplest model: *finite state machines* or *finite automata*.

Finite automata are good models for computers with an *extremely limited amount of memory*. They are nonetheless useful for many things!

Example: A Controller for Automatic Door

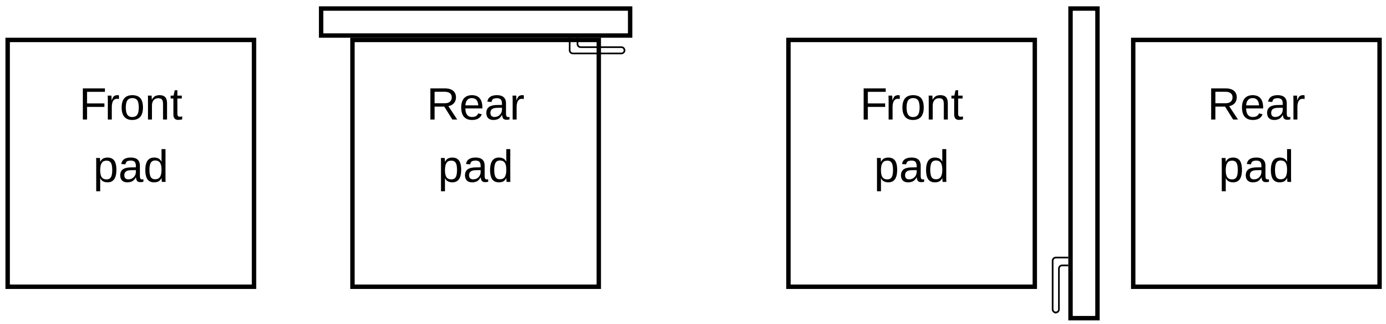


Correct behaviour:

- If a person is on the front pad, the door should open.
- It should remain open long enough for the person to pass all the way through.
- The door should not strike someone standing behind it (i.e. on the rear pad) as it opens.

Example: A Controller for Automatic Door (cont'd)

Two states: OPEN, CLOSED



OPEN *CLOSED*

Four "input conditions":

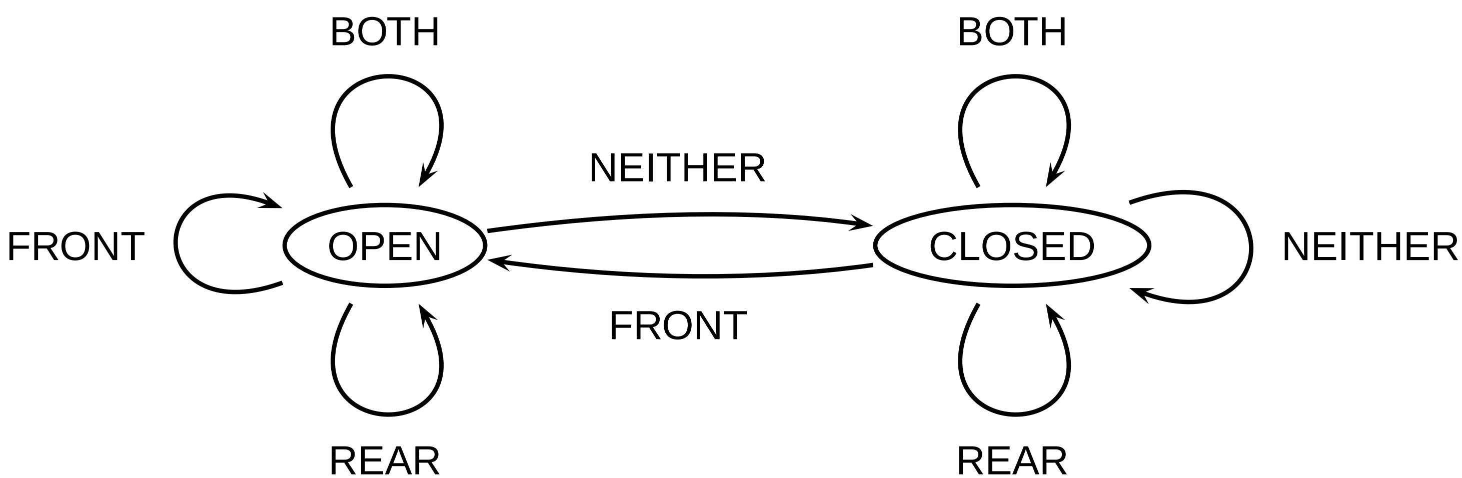
- FRONT: someone standing on front pad only
- REAR: someone standing on rear pad only
- BOTH: people standing on both pads
- NEITHER: no one standing on either pad

Example: A Controller for Automatic Door (cont'd)

State transition table:

	NEITHER	FRONT	REAR	BOTH
CLOSED	CLOSED	OPEN	CLOSED	CLOSED
OPEN	CLOSED	OPEN	OPEN	OPEN

Note: the table can be presented as a (state-transition) graph



The controller is a rudimentary computer that has just a single bit of memory (for recording state information).

It is an example of a *finite automaton* (or a *finite-state machine*).

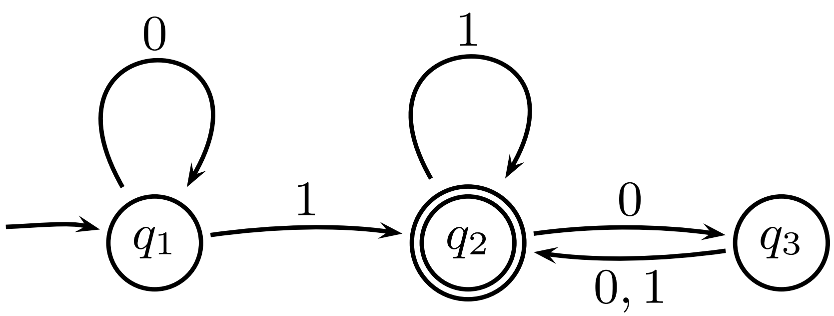
Other examples: controllers of dishwashers, electronic thermostats, parts of digital watches and calculators, etc.

References

1. M. D. Davis, R. Sigal and E. J. Weyuker. *Computability, Complexity and Languages*. Academic Press, 2nd edition, 1994. ISBN: 0122063821.
2. J. E. Hopcroft, R. Motwani and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 2nd edition, 2001.
3. D. C. Kozen. *Automata and Computability*. Springer-Verlag, 3rd printing, 2000.
4. H. R. Lewis and C. Papadimitriou. *Elements of the Theory of Computation*. Prentice-Hall, 2nd edition, 1997.
5. M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, January 1997.
6. T. A. Sudkamp. *Languages and Machines*. Addison-Wesley, 2nd edition, 1996. ISBN: 0201821362.

Deterministic Finite Automata (DFA)

Example: a finite automaton M_1



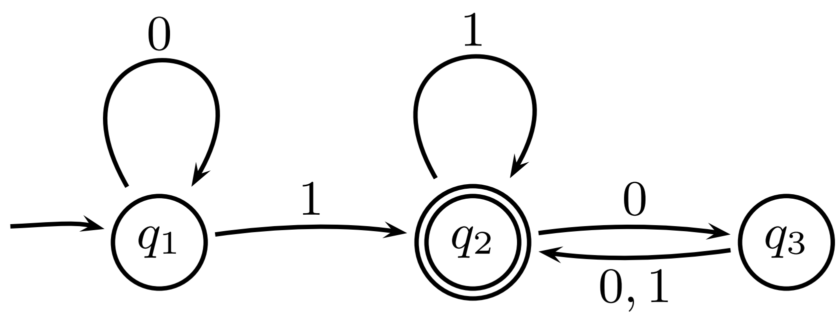
Key Features:

- There are only finitely different *states* a finite automaton can be in.

  The states in M_1 (= vertices of the graph) are q_1, q_2 and q_3.

- We do not care about the internal structure of automaton states. All we care about is which *transitions* the automaton can make between states.

- A symbol from some finite alphabet Σ is associated with each transition: we think of elements of Σ as *input symbols*. The alphabet of M_1 is $\{ 0, 1 \}$.



- Thus all possible transitions can be specified by a *finite directed graph with Σ-labelled edges*.

  E.g. At state q_2, M_1 can

  - input 0 and enter state q_3 i.e. $q_2 \xrightarrow{0} q_3$, or
  - input 1 and remain in state q_2 i.e. $q_2 \xrightarrow{1} q_2$.

- There is a distinguished *start state*. In the graph, the start state is indicated by an arrow pointing at it from nowhere. The start state of M_1 is q_1.

- The states are partitioned into *accepting* states (or final states) and *non-accepting* states.

  An accepting state is indicated by a (double) circle. The accepting state of M_1 is q_2.

Why designate certain states *accepting*?

Notation. We write Σ^* as the set of all *strings* (or words) over Σ i.e.

$$\Sigma^* \stackrel{\text{def}}{=} \{ a_1 \cdots a_k : a_i \in \Sigma, k \geq 0 \}.$$

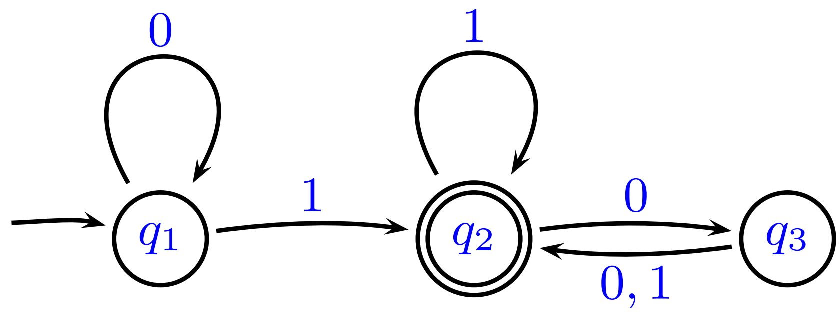
A *language* L is just a set of strings over Σ i.e. $L \subseteq \Sigma^*$.

> We use a finite automaton to recognize whether or not a string $u \in \Sigma^*$ is in a particular language (= subset of Σ^*).

Given u, we begin in the start state, and traverse the state-transition graph, using up the symbols in u in the correct order, reading from left to right.

If we can consume all the symbols u in this way and reach an accepting state, then u is in the *language accepted* (or recognized) by the particular automaton; otherwise u is not in the language.

M_1 Revisited



What is the language accepted by M_1?

Answer: all binary strings that contain at least one 1, and an even number of 0s follow the last 1.

When M_1 receives an input string (say) 1101, it processes the string and produces either a “yes” (meaning: the input is accepted) or “no” result.

Beginning at the start state, M_1 receives the symbols from the input string one by one from left to right; after reading each symbol, M_1 moves from one state to another along the transition labelled by that symbol.

After the last symbol is read, M_1 returns “yes” if it is at a final state, and “no” otherwise.

E.g. In processing 1101, M_1 goes through the states q_1, q_2, q_2, q_3, q_2, and returns “yes”.

Definition: Deterministic Finite Automaton (DFA)

A *deterministic finite automaton (DFA)* is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

(i) Q is a finite set called the *states*

(ii) Σ is a finite set called the *alphabet*

(iii) $\delta : Q \times \Sigma \longrightarrow Q$ is the *transition function*

(iv) $q_0 \in Q$ is the *start state*

(v) $F \subseteq Q$ is the *set of accept states* (or *final states*).

We write $q \xrightarrow{a} q'$ to mean $\delta(q, a) = q'$, which we read as "there is an a-*transition* from q to q'".

State-transition graph of a DFA

Equivalently we can represent a DFA by its *state-transition graph*:

- the vertices are just the states
- Σ-labelled edges are the transitions.

Notation:

- The start state is indicated by an arrow pointing at it from nowhere.
- A final state is indicated by a (double) circle; the labelled arrows from one state to another are called *state-transitions*.

Note:

Such a state-transition graph represents a DFA, if for all $a \in \Sigma$ there is exactly one outgoing a-labelled edge from each vertex (state).

Definition: Language accepted by M, $L(M)$

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. $L(M)$, the *language recognized (or accepted) by the DFA* M, consists of all strings $w = a_1 a_2 \cdots a_n$ over Σ satisfying $q_0 \xrightarrow{w}^* q$ where q is a final state. Here

$$q_0 \xrightarrow{w}^* q$$

means that there exist states $q_1, \cdots, q_{n-1}, q_n = q$ (not necessarily all distinct) such that there are transitions of the form

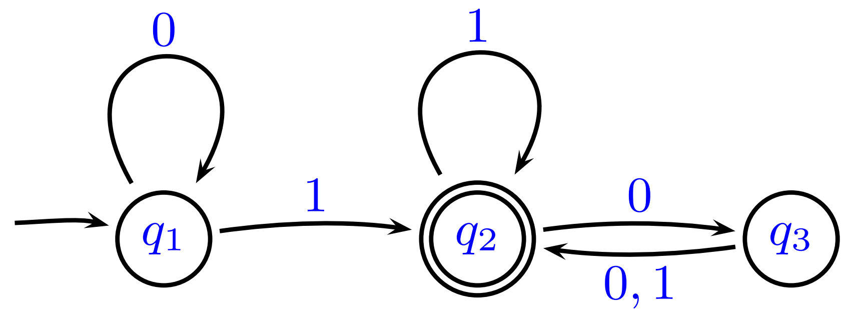
$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n = q$$

Note

- case $n = 0$: $q \xrightarrow{\epsilon}^* q'$ iff $q = q'$
- case $n = 1$: $q \xrightarrow{a}^* q'$ iff $q \xrightarrow{a} q'$

A language is called *regular* if some DFA recognizes it.

Example: M_1 Revisited



Formally $M_1 = (Q, \Sigma, \delta, q_1, F)$ where

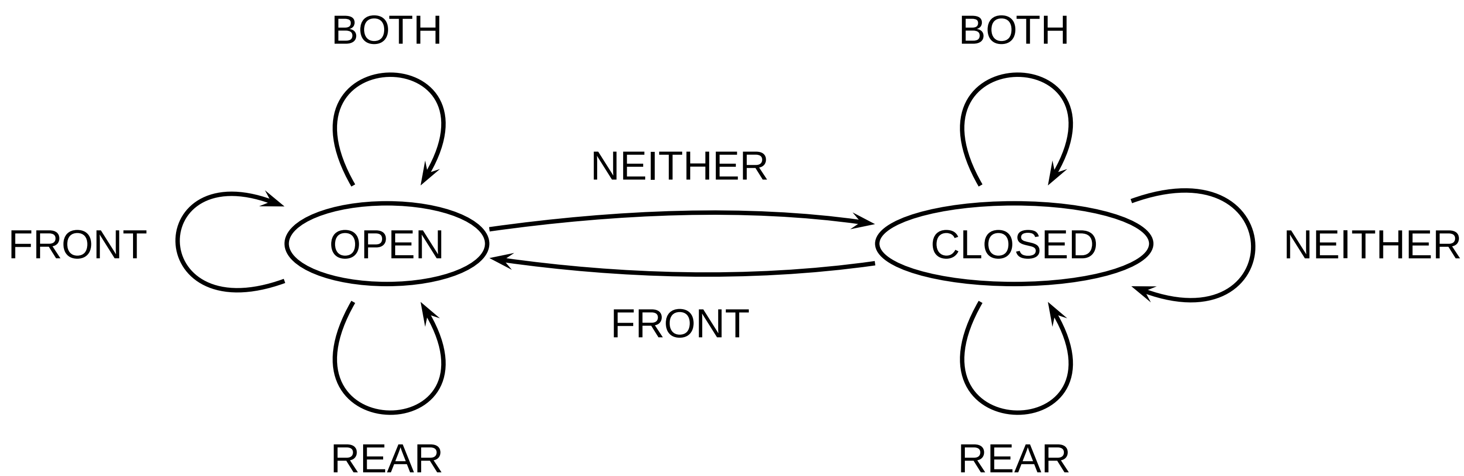
- $Q = \{ q_1, q_2, q_3 \}$
- $\Sigma = \{ 0, 1 \}$
- q_1 is the start state; $F = \{ q_2 \}$
- δ is given by

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

$L(M_1)$ is the set of all binary strings that contain at least one 1, and an even number of 0s follow the last 1.

More examples: automatic door controller

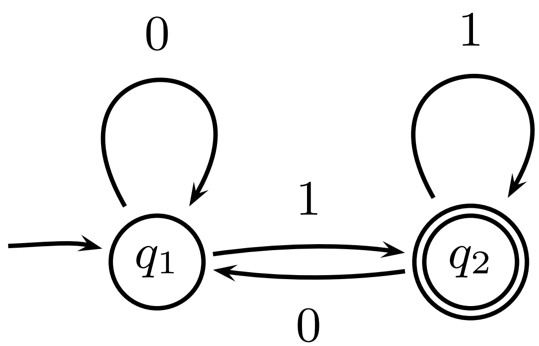
Is the controller for the automatic door a DFA?



So far, the control has no designated start state, or designated accepting states, but otherwise it is a DFA, with

- state set: $\{\text{OPEN}, \text{CLOSED}\}$
- input alphabet: $\{\text{FRONT}, \text{REAR}, \text{BOTH}, \text{NEITHER}\}$
- transition function: as determined by the state transition table given earlier

More examples: M_2



$$M_2 = (\{q_1, q_2\}, \{0, 1\}, \delta, q_1, \{q_2\})$$

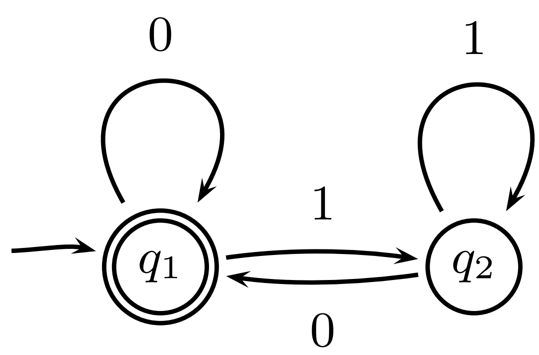
where δ is given by

	0	1
q_1	q_1	q_2
q_2	q_1	q_2

The language recognized by M_2 is

$L(M_2) = \{w \in \{0,1\}^* :$ w ends in a $1\}$

More examples: M_3



$$M_3 = (\{q_1, q_2\}, \{0, 1\}, \delta, q_1, \{q_1\})$$

where δ is given by

	0	1
q_1	q_1	q_2
q_2	q_1	q_2

The language recognized by M_3 is

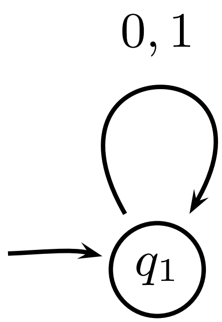
$$L(M_3) = \{w \in \{0,1\}^* : w \text{ ends in a } 0\} \cup \{\epsilon\}$$

Designing DFAs over input alphabet $\{0, 1\}$

Find M such that $L(M) = \emptyset$.

Designing DFAs over input alphabet $\{0, 1\}$

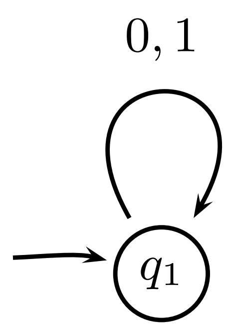
Find M such that $L(M) = \emptyset$.



no final state!

Designing DFAs over input alphabet $\{0, 1\}$

Find M such that $L(M) = \emptyset$.

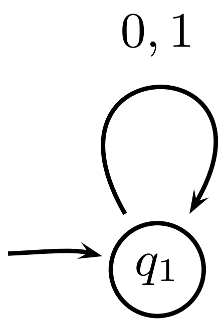


no final state!

Find M such that $L(M) = \{\epsilon, 0\}$.

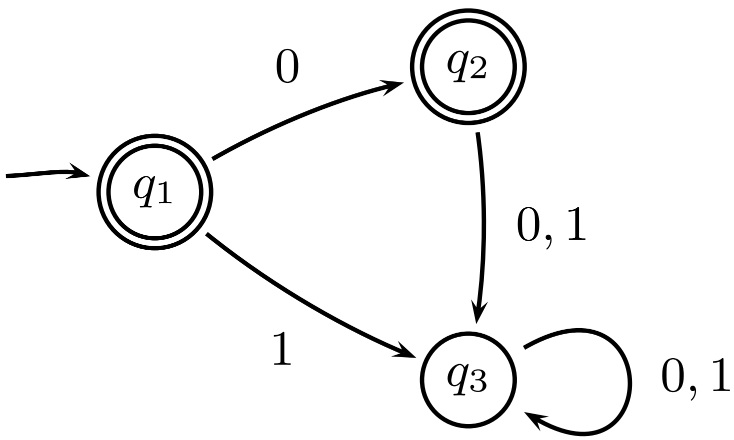
Designing DFAs over input alphabet $\{0, 1\}$

Find M such that $L(M) = \emptyset$.



no final state!

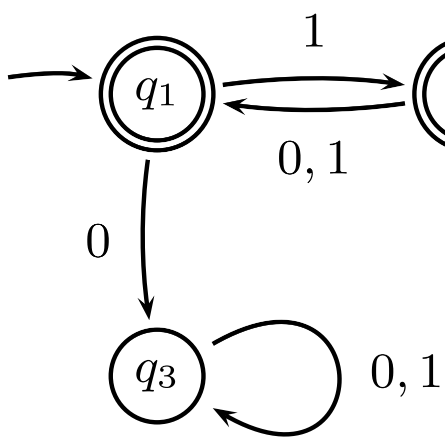
Find M such that $L(M) = \{\epsilon, 0\}$.



Designing DFAs over input alphabet $\{0, 1\}$

Find M such that $L(M) = \{w \in \{0, 1\}^*$: every odd position of w is a $1\}$.

E.g. $L(M) = \{\epsilon, 1, 10, 11, 101, 111, 1010, 1011, 1110, 1111, \ldots\}$



The Regular Operations: Union, Concatenation and Star

Let A and B be languages. Define

- **Union**: $A \cup B = \{ x : x \in A$ or $x \in B \}$
- **Concatenation**: $A \cdot B = \{ xy : x \in A$ and $y \in B \}$
- **Star**: $A^* = \{ x_1 x_2 \cdots x_k : k \geq 0$ and each $x_i \in A \}$.

Note: ϵ (the empty string) is in A^* (the case of $k = 0$)

Example. Take $A = \{ good, bad \}$ and $B = \{ boy, girl \}$.

$A \cdot B = \{ goodboy, goodgirl, badboy, badgirl \}$

$A^* =$

$\{ \epsilon, good, bad, goodgood, goodbad, badgood, badbad, goodgoodgood, \cdots \}$.

Informally $A^* = \{ \epsilon \} \ \cup \ A \ \cup \ (A \cdot A) \ \cup \ (A \cdot A \cdot A) \ \cup \ \cdots$.

The Product Construction

Theorem Regular languages are closed under union i.e. if A_1 and A_2 are regular languages, so is $A_1 \cup A_2$.

Proof. Simulate M_1 and M_2 *simultaneously*!

Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1, and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2.

We construct $M = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1 \cup A_2$:

- $Q = Q_1 \times Q_2$ (= $\{ (r_1, r_2) : r_1 \in Q_1, r_2 \in Q_2 \}$)
- $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
- $q_0 = (q_1, q_2)$
- $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$ (= $\{ (r_1, r_2) : r_1 \in F_1 \vee r_2 \in F_2 \}$)

We first show $L(M) \subseteq L(M_1) \cup L(M_2)$:

Take $w = a_1 \cdots a_n \in L(M)$. By definition, for some $q_0^1 = q_1, q_1^1, \cdots, q_n^1 \in Q_1$, for some $q_0^2 = q_2, q_1^2, \cdots, q_n^2 \in Q_2$, we have M-transitions

$$(q_0^1, q_0^2) \xrightarrow{a_1} (q_1^1, q_1^2) \xrightarrow{a_2} \cdots\cdots \xrightarrow{a_n} (q_n^1, q_n^2) \tag{1}$$

where $q_n^1 \in F_1$ or $q_n^2 \in F_2$. Suppose the former. Unpacking (1), we have M_1-transitions

$$q_0^1 \xrightarrow{a_1} q_1^1 \xrightarrow{a_2} \cdots\cdots \xrightarrow{a_n} q_n^1 \tag{2}$$

I.e. $q_0^1 (= q_1) \xrightarrow{w}{}^* q_n^1 \in F_1$. Hence $w \in L(M_1) \subseteq L(M_1) \cup L(M_2)$.

Next we show $L(M_1) \subseteq L(M)$ (argument for $L(M_2) \subseteq L(M)$ is similar):

Take $w = a_1 \cdots a_n \in L(M_1)$. By definition, for some $q_0^1 = q_1, q_1^1, \cdots, q_n^1 \in Q_1$, with $q_n^1 \in F_1$, we have

$$q_0^1 \xrightarrow{a_1} q_1^1 \xrightarrow{a_2} \cdots\cdots \xrightarrow{a_n} q_n^1 \tag{3}$$

Now since δ_2 is a function, for any $q \in Q_2, a \in \Sigma$, there is a q' s.t. $q \xrightarrow{a} q'$. Hence, there are $q_0^2 = q_2, q_1^2, \cdots, q_n^2 \in Q_2$ s.t.

$$(q_0^1, q_0^2) \xrightarrow{a_1} (q_1^1, q_1^2) \xrightarrow{a_2} \cdots\cdots \xrightarrow{a_n} (q_n^1, q_n^2) \tag{4}$$

are M-transitions. Since $(q_n^1, q_n^2) \in F_1 \times Q_2 \subseteq F$, we have $w \in L(M)$.

$\square$

Remark. Closure under union can be proved (quite simply) using NFAs.

Motivation: Nondeterministic Finite Automata

> **Theorem** Regular languages are closed under concatenation i.e. if A_1 and A_2 are regular languages, so is $A_1 \cdot A_2$.

Proof attempt:

Let $L(M_i) = A_i$. Aim to construct M that accepts w iff w can be broken into w_1 and w_2 (so that $w = w_1 w_2$) whereby M_1 accepts w_1 and M_2 accepts w_2.

But M does not know where to break w into two!

This motivates the introduction of *non-deterministic* finite automata.