Finite State Automata Design

Nicolò Felicioni[1]

Dipartimento di Elettronica e Informazione
Politecnico di Milano

nicolo . felicioni @ polimi . it

March 1, 2021

[1] Mostly based on Nicholas Mainardi and Alessandro Barenghi's material, enriched by few additional examples.

Finite State Automata

What are FSAs?

- Finite State Automata (Automi a stati finiti AF) are the simplest among abstract computing models
- They are still able to describe a good deal of useful systems
- They are characterized by a finite memory capacity
- In a sense, every real world computing machine is a FSA (as it only has a limited number of memory cells)
- However modeling it as such is not a good idea (too many states, roughly $2^{2^{43}} \simeq 10^{300000000000}$)

Formalization

- A recognizer FSA is formally defined as a quintuple $(\mathbf{Q}, \mathbf{I}, \delta, q_0, \mathbf{F})$, where:
  - $\mathbf{Q}$, is the set of states of the automata
  - $\mathbf{I}$ is the alphabet of the input string which will be checked
  - $\delta : \mathbf{Q} \times \mathbf{I} \mapsto \mathbf{Q}$ the transition function
  - $q_0 \in \mathbf{Q}$ the (unique) initial state from where the automaton starts
  - $\mathbf{F} \subseteq \mathbf{Q}$ the set of final accepting states of the automaton

Regular expressions cheat sheet

A small RE cheat sheet

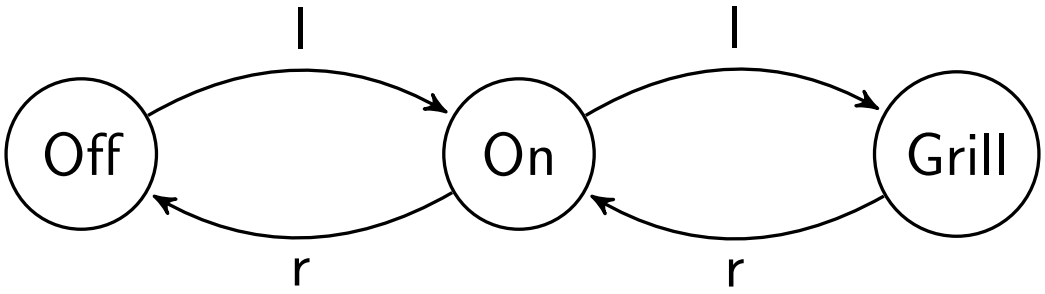
- Symbols belonging to the input alphabet are usually lowercase
- `s.t` : `s` concatenated with `t`
- `r` | `s` : either `r` or `s`
- Parentheses force a precedence when reading the regexpr
- $\mathtt{s}^*$: an arbitrary number of occurrences of `s` (Kleene's Star)
- $\mathtt{s}^+$: one or more occurrences of `s` (equivalent to $\mathtt{s}(\mathtt{s})^*$)
- $\mathtt{s}^?$: `s` is optional $\rightarrow$ zero or one occurrence of `s` (equivalent to $\mathtt{s} \mid \epsilon$)

Introduction

An oven knob

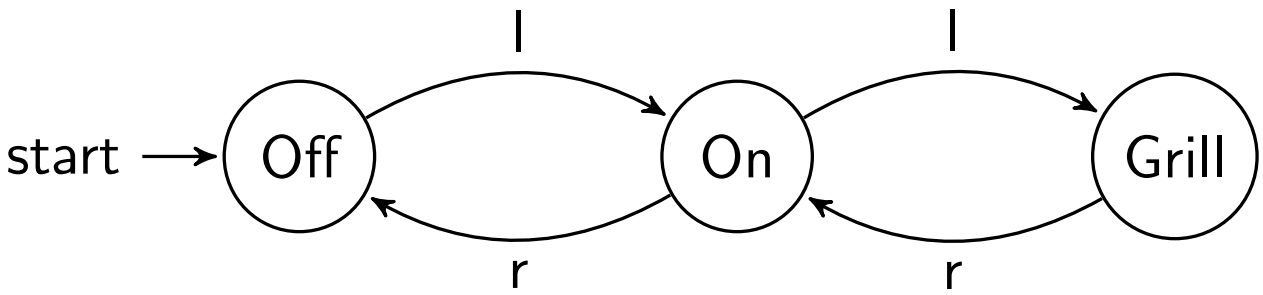
- We will start looking at the design for a simple automaton representing an oven knob
- The oven may be *On*, *Off* or on *Grill* position.
- The knob turns left and right: left raises the temperature, right lowers it.

A first attempt



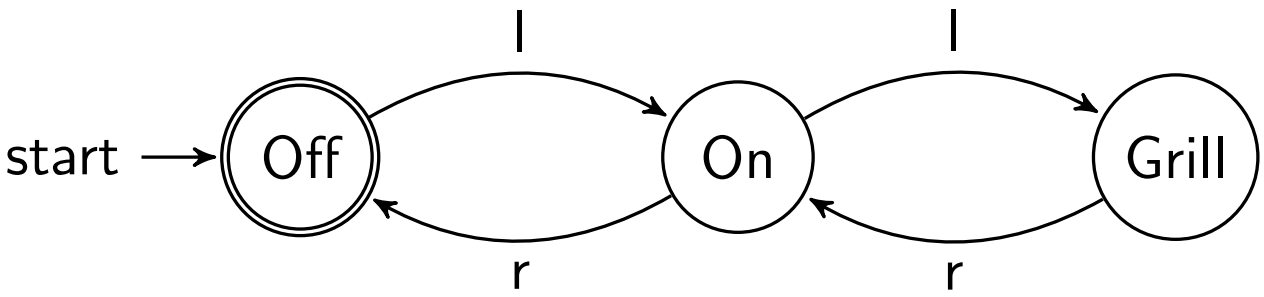
But this oven starts in an undefined state...

A first attempt



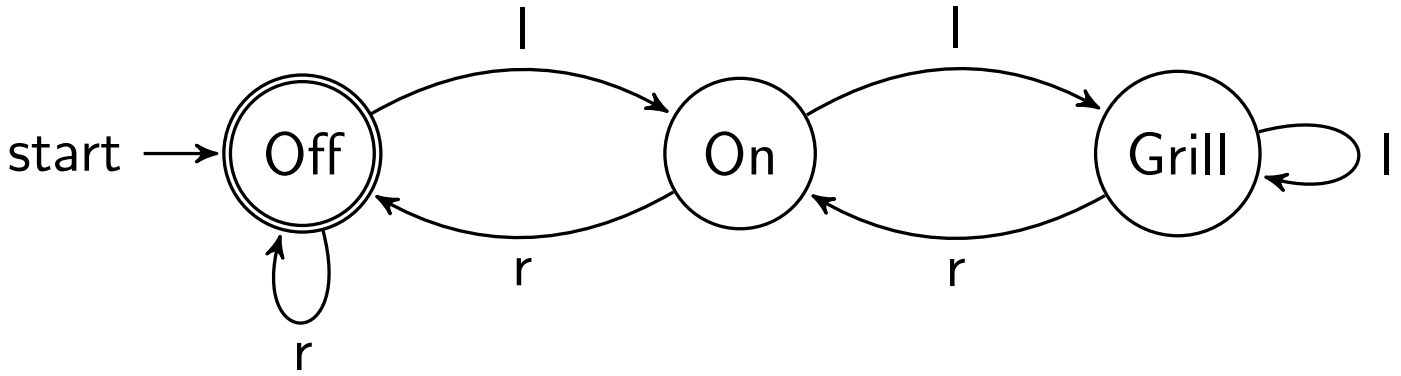
This one's better, but we'd like to accept only when we remembered to turn it off at the end of the cooking...

A first attempt



Ok, but what if the knob is turned further left (resp. right) when it's already on the "Grill" (resp. "Off") position?

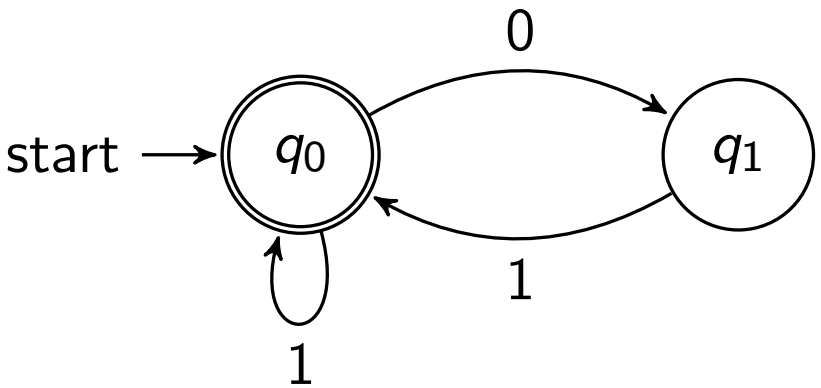
A first attempt



Great. What sequences of actions (language strings) does this recognize? $L = (r \mid l(l(l)^* r)^* r)^* = (r \mid l(l^+ r)^* r)^*$

A first recognizer

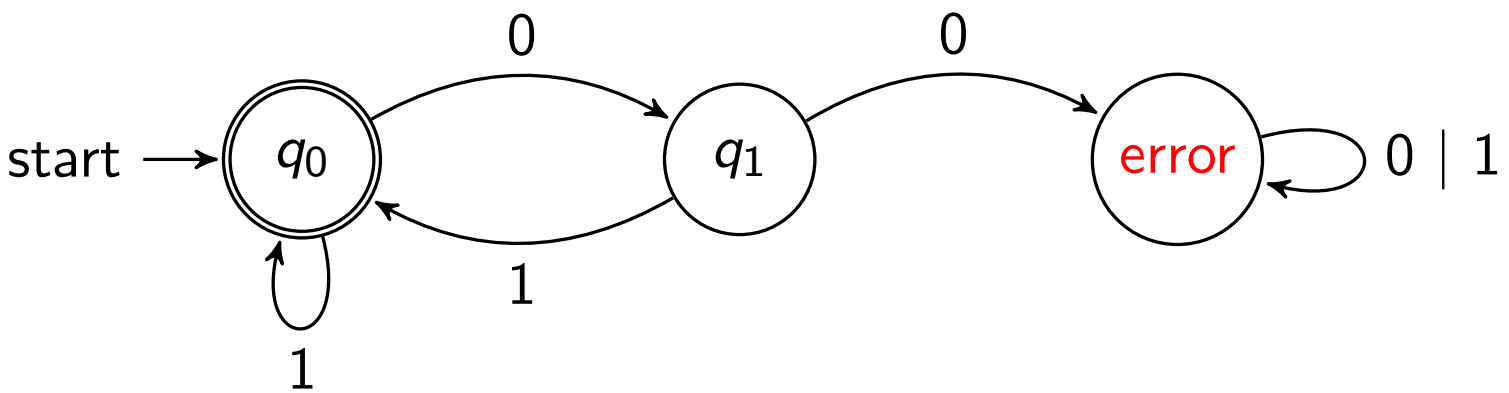
- Let's try the other way around: design a FSA recognizing a specific language
- Example: $L = (01 \mid 1)^*$, that is the language where every zero is followed by at least a one (i.e. there cannot be two consecutive zeros)



- $\delta(q_1, 0)$ is undefined here!

The error state

- The previous FSA does not explicitly represent error states
- As this may result in issues when performing operation among automata, we'll add it



- The $\delta(\cdot, \cdot)$ function is now complete (safer to perform complement).

Complement

How-To

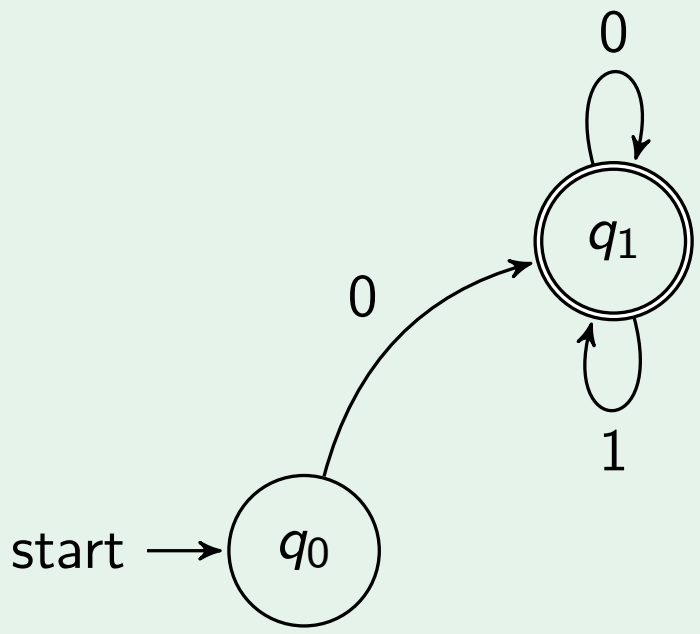
- Given a recognizer FSA for the language L, the one recognizing the complement of the language L^C can be obtained flipping the termination condition
- Basically: all the accepting states become non accepting and vice-versa
- Take care: the error state is nothing more than a common non accepting state and must be included in the process.
- A safe way to take it into account is to complete the automaton before building the complementary one

Complement

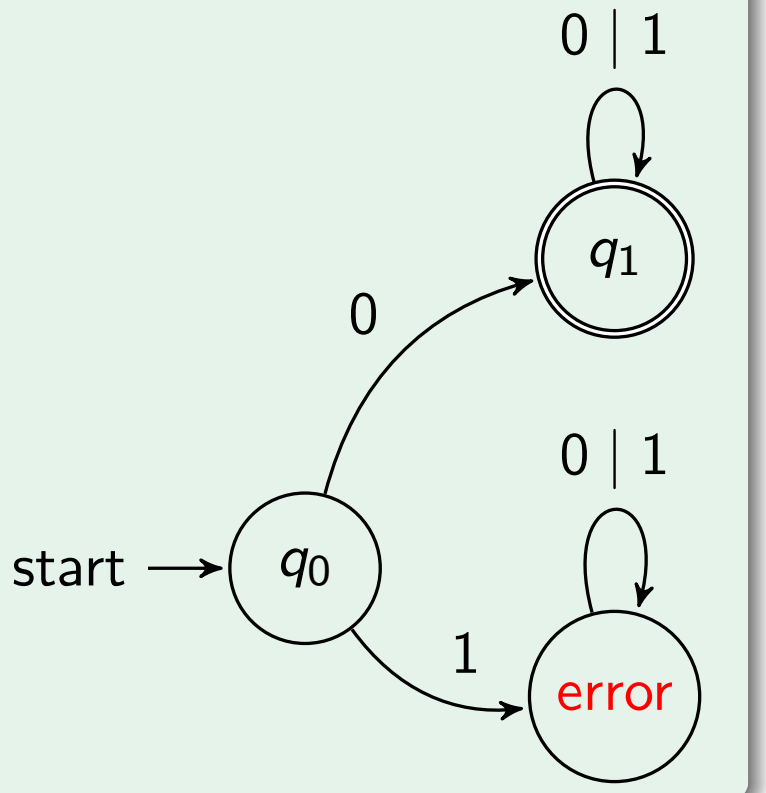
- Let's take as an example the recognizer for $L = 0(0|1)^*$

The recognizer looks like this :

Partial $\delta(\cdot, \cdot)$



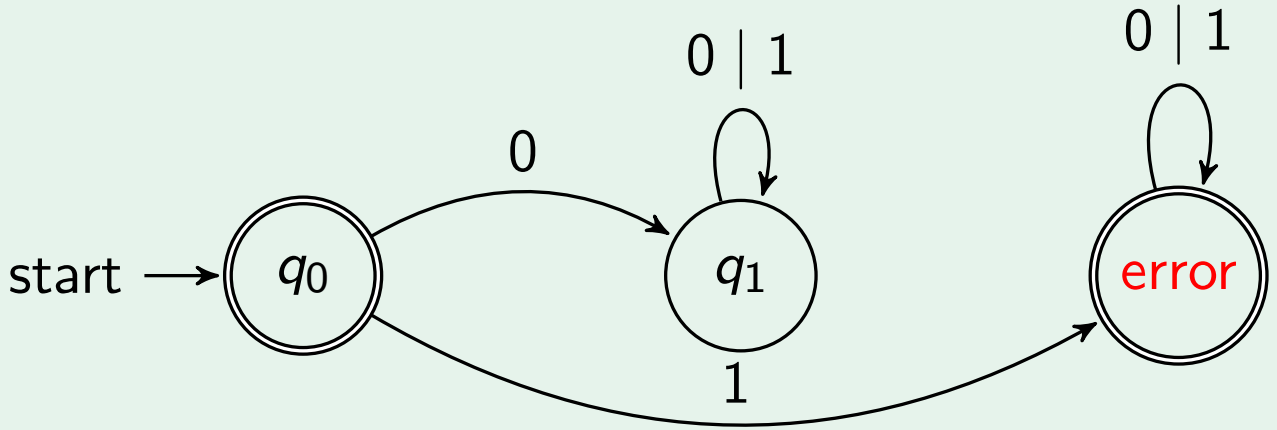
Complete $\delta(\cdot, \cdot)$



Complement

- The recogniser for L^C is thus :

Recogniser for $(0|1)^* \setminus L$

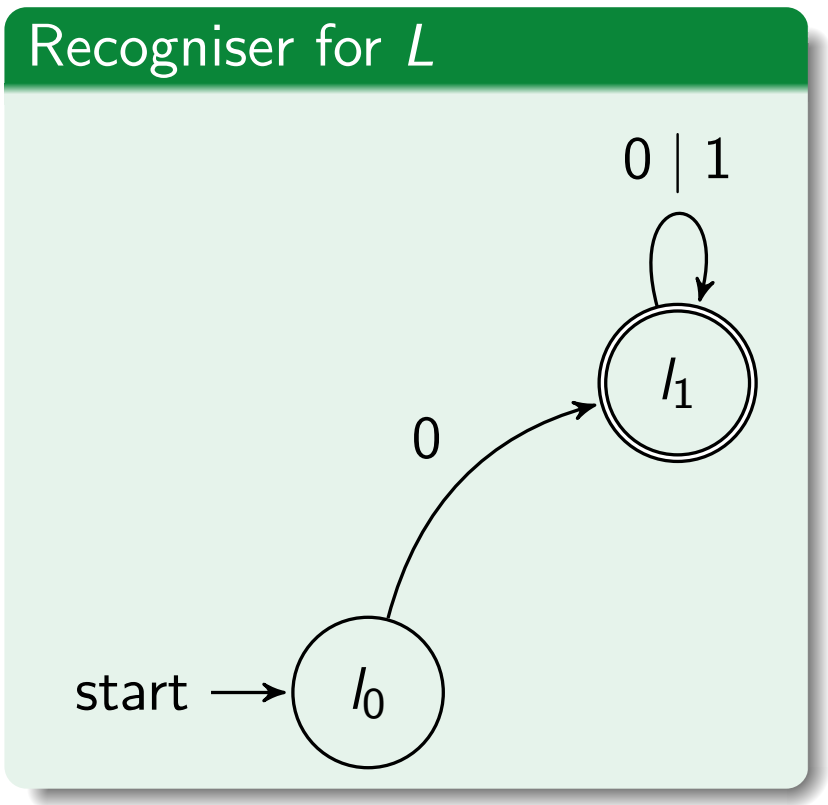


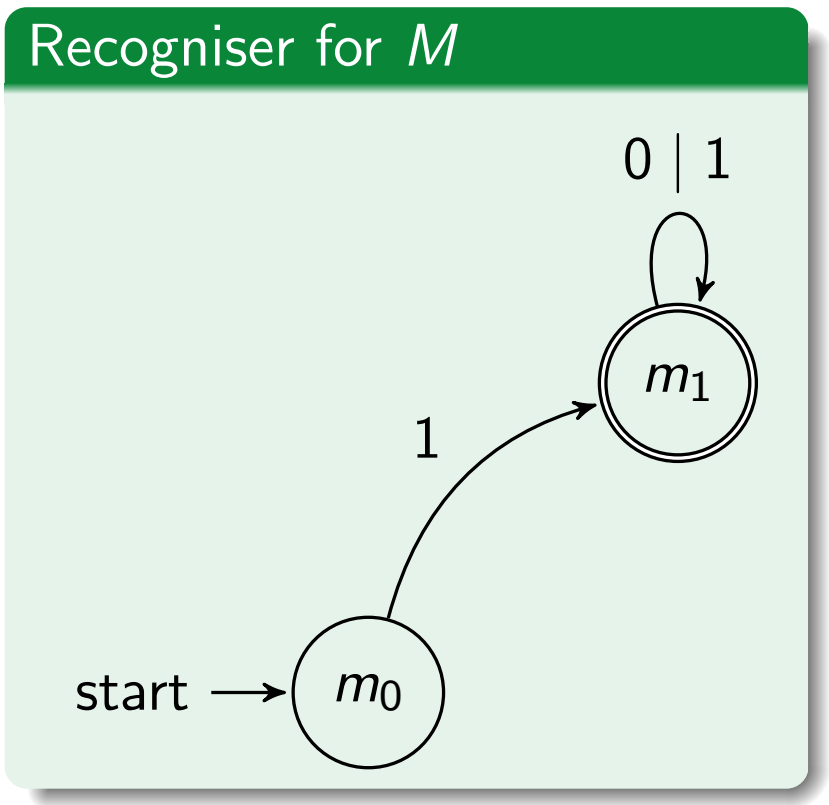
Intersection

- The recogniser automaton for the intersection of two languages $L \cap M$ can be built starting from the ones recognising them
- The steps to be followed are :
  1. Build the state set as the Cartesian product of the state sets (hint: label the states with the combination of the two original labels, that is $\langle l_i, m_i \rangle$)
  2. Set the initial state to the one obtained via the Cartesian product of l_0 and m_0
  3. Start deriving the δ function from the initial state: simply check the original deltas and choose the destination state obtained as the product of the destinations. If at least one of the two original deltas is undefined, than δ is undefined too.
  4. Mark as final only the states obtained as the product of two final states: $\langle l_i, m_i \rangle \in F$ if and only if $l_i \in F_l \wedge m_i \in F_m$

Example intersection

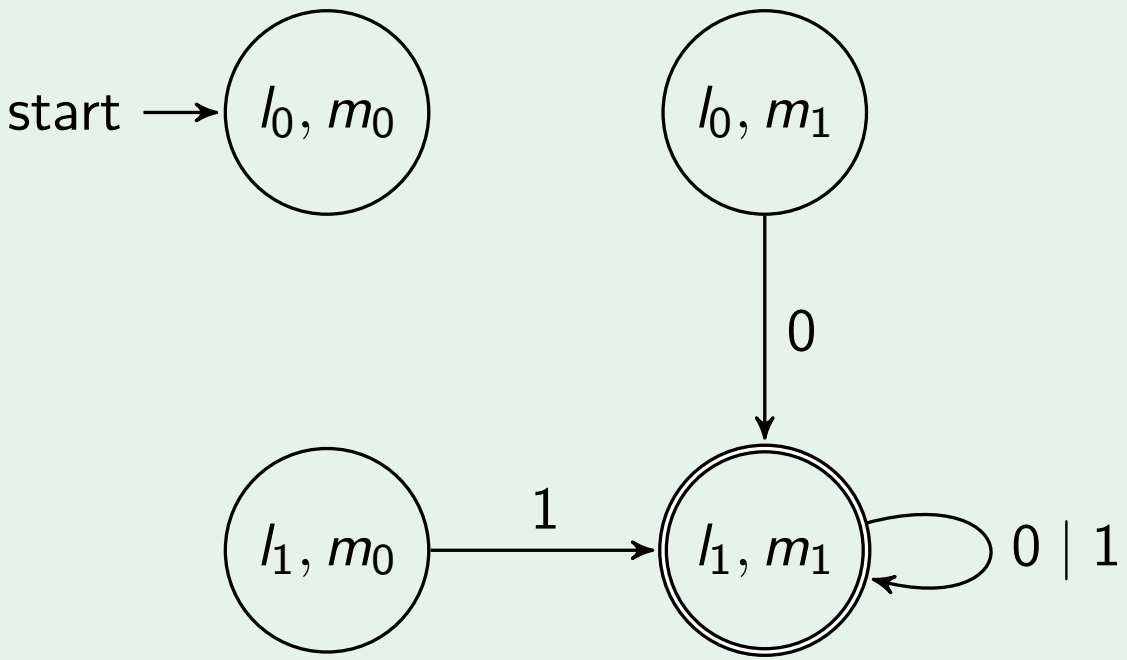
- We want to obtain a recogniser for $L \cap M$





Intersection Automaton

Automaton for $\mathbf{L} \cap \mathbf{M} = \emptyset$ including unreachable states)



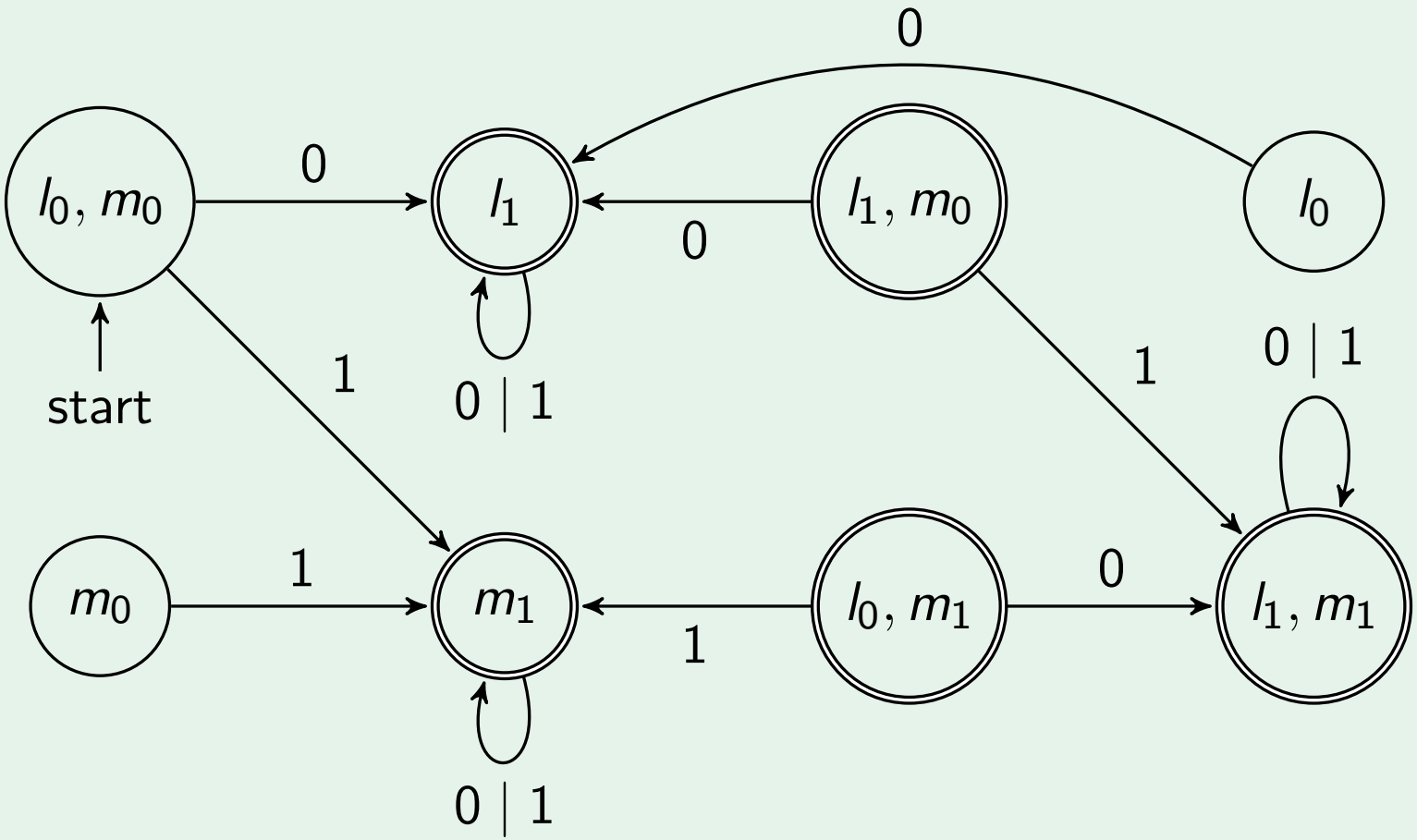
Union

Given the automata $A_l = \langle \mathbf{Q_l}, \mathbf{I}, \delta_l, l_0, \mathbf{F_l} \rangle$ and $A_m = \langle \mathbf{Q_m}, \mathbf{I}, \delta_m, m_0, \mathbf{F_m} \rangle$ for, respectively, languages L and M, the recogniser automaton $A = \langle \mathbf{Q}, \mathbf{I}, \delta, q_0, \mathbf{F} \rangle$ for $L \cup M$:

- $\mathbf{Q} = \mathbf{Q_l} \times \mathbf{Q_m} \cup \mathbf{Q_l} \cup \mathbf{Q_m}$, that is the Cartesian product of the state sets is added to the state sets of the original automata
- $\forall l_i \in \mathbf{Q_l}, m_j \in \mathbf{Q_m}, a \in \mathbf{I}$, $\delta(\langle l_i, m_j \rangle, a)$ is equal to:
  - $\langle \delta_l(l_i, a), \delta_m(m_j, a) \rangle$ if $\delta_l(l_i, a) \neq \perp \wedge \delta_m(m_j, a) \neq \perp$
  - $\delta_l(l_i, a)$ if $\delta_l(l_i, a) \neq \perp \wedge \delta_m(m_j, a) = \perp$
  - $\delta_m(m_j, a)$ if $\delta_l(l_i, a) = \perp \wedge \delta_m(m_j, a) \neq \perp$
  - $\perp$ if $\delta_l(l_i, a) = \perp \wedge \delta_m(m_j, a) = \perp$
- $q_0 = \langle l_0, m_0 \rangle$
- $\mathbf{F} = \mathbf{F_l} \times \mathbf{Q_m} \cup \mathbf{F_m} \times \mathbf{Q_l} \cup \mathbf{F_l} \cup \mathbf{F_m}$, that is a state is marked as final if it is the product of at least one final state of the original automata

Union Automaton

Automaton for $\mathbf{L} \cup \mathbf{M} = \{\mathbf{0}, \mathbf{1}\}^+$ (including unreachable states)



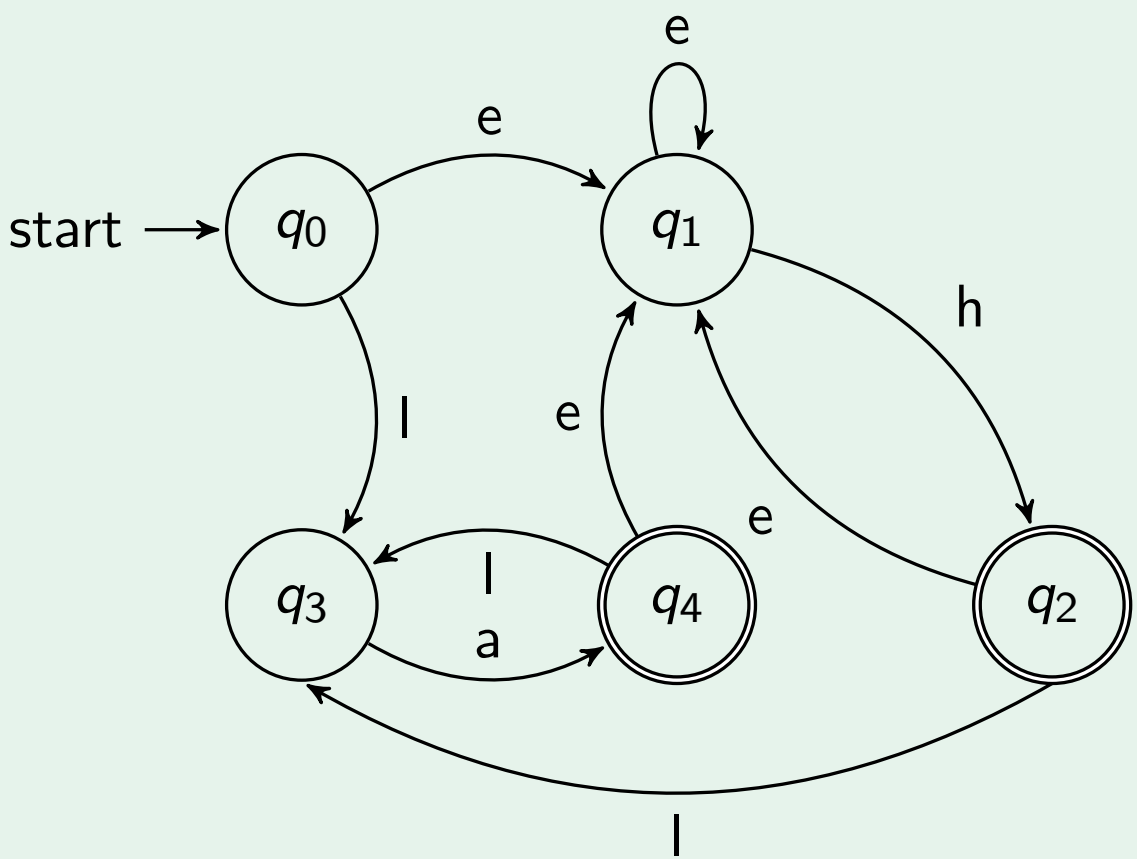
Union: Alternative Strategy

- It is possible to use complement and intersection to compute the recognizer automaton for $L \cup M$
- The key idea is to exploit De Morgan's Law applied to set operations: $\neg(a \cup b) = (\neg a) \cap (\neg b)$
- It is thus possible to derive $a \cup b = \neg(\neg(a \cup b)) = \neg((\neg a) \cap (\neg b))$
- Therefore, by applying in sequence two complements, an intersection and another complement we obtain the union automaton

Vasco Rossi's lyrics

Vasco Rossi's lyrics clearly belong to the language $L = (e^+h|la)^+$.
Therefore, we can recognize Vasco's lyric with a FSA:

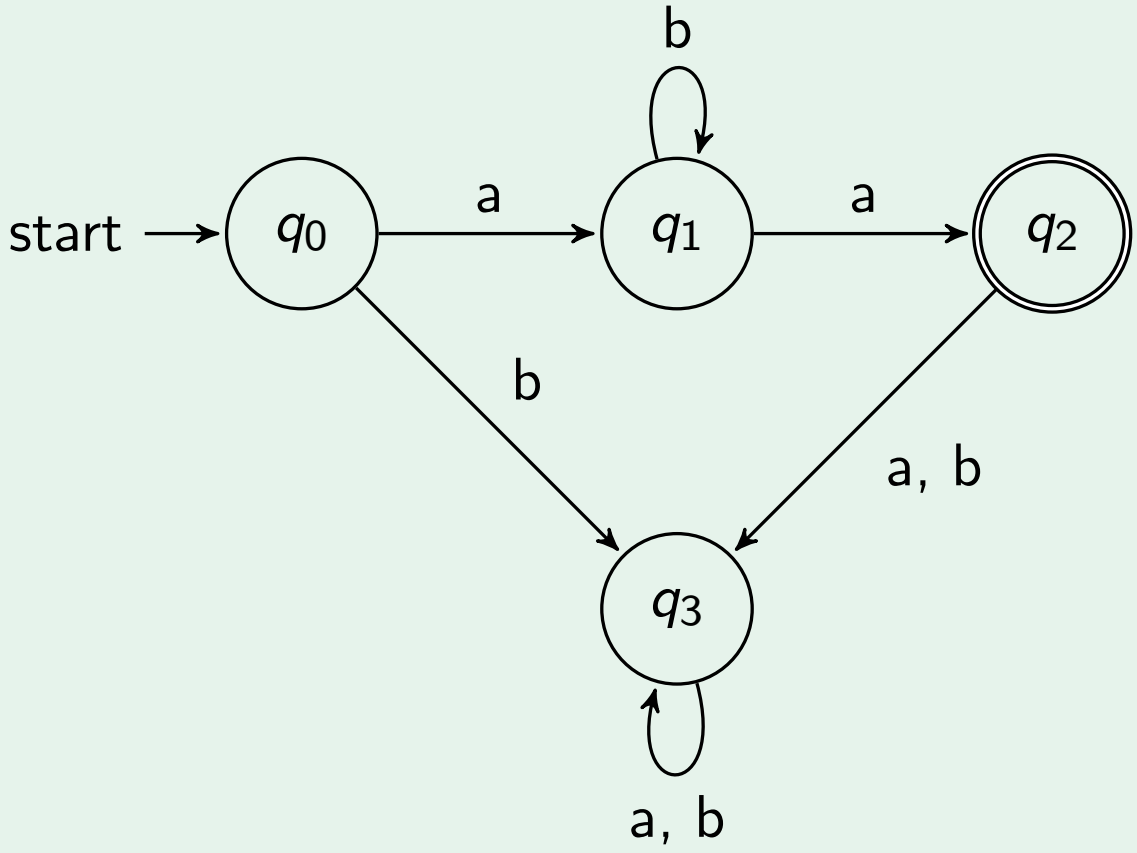
FSA Recognizer



From FSA to its language

Given the following FSA, obtain its language.

FSA Recognizer



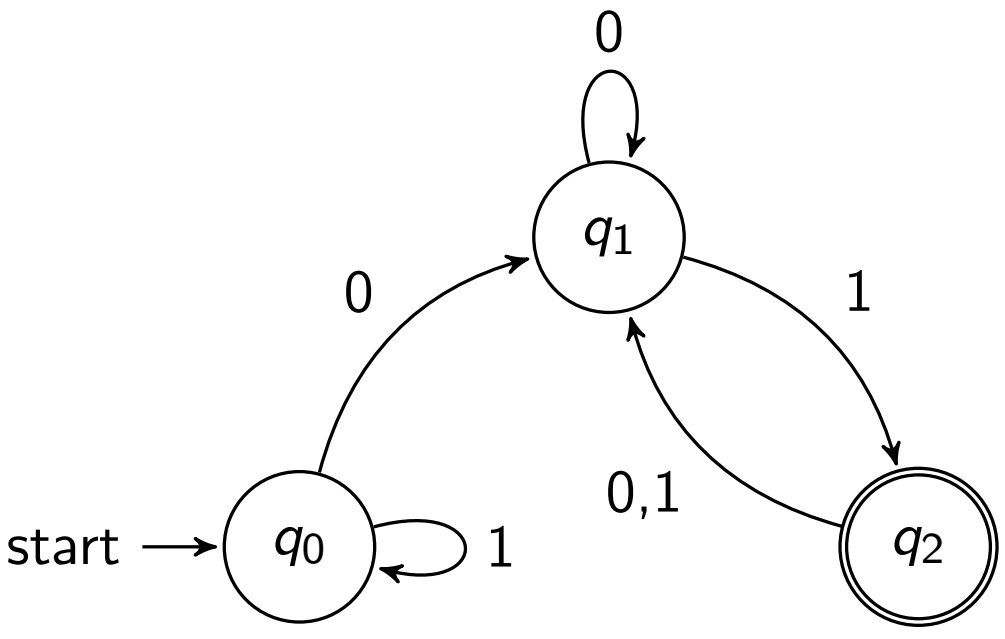
From FSA to its language

Language

$L = ab^*a$. The q_3 state is a *sink* state.

Proving Properties on an FSA

- The following automaton recognizes $L = (0|1)^*01^{2k+1}$.



- Even if it is possible to intuitively check that this is right, we need to provide a formal proof
- The most straightforward way to prove it is by induction on the steps performed

Mathematical Induction

- The idea is to prove by induction (on the steps performed by the automaton):
  - That all the strings in the language lead the computation in an accepting state
  - That all the strings not in the language lead the computation to a non accepting state
- To this end, we partition the set of possible strings I^* in as many sets as the number of states of the complete automaton, associating each set to a state of the automaton:
  - q_0 Strings without a 0, i.e. $L_1 = 1^*$ $(\notin L)$
  - q_1 Strings with at least a 0 and ending with an even number of 1, i.e. $L_2 = \{(0|1)^*01^{2k}, k \geq 0\}$ $(\notin L)$
  - q_2 Strings with at least a 0 and ending with an odd number of 1, i.e. $L_3 = \{(0|1)^*01^{2k+1}, k \geq 0\}$ $(\in L)$

Mathematical Induction

Building hypotheses and checking base cases

- Our purpose is thus to prove that:
  1. $\forall s \in L_1,\ \delta^*(q_0, s) = q_0$
  2. $\forall s \in L_2,\ \delta^*(q_0, s) = q_1$
  3. $\forall s \in L_3,\ \delta^*(q_0, s) = q_2$
- These three propositions will work for us as induction hypotheses
- Since we are performing the induction on the steps of the automaton, the *base cases* are represented by the shortest strings for each terminating character and from each class
- Let's start from checking the base cases:
  1. $\delta(q_0, \epsilon) = q_0$ and $\delta(q_0, 1) = q_0$? Yes.
  2. $\delta(q_0, 011) = q_1$ and $\delta(q_0, 0) = q_1$? Yes.
  3. $\delta(q_0, 01) = q_2$? Yes.

Mathematical Induction

Proofs

1. $\forall s \in L_1 : \delta^*(q_0, s = s'.i) = \delta(\delta^*(q_0, s'), i) = \delta(\delta^*(q_0, s'), 1) = \delta(q_0, 1)$, since $s' \in L_1$. Then, $\delta(q_0, 1) = q_0$, thus $\forall s \in L_1$, $\delta^*(q_0, s) = q_0$ is true
2. $\forall s \in L_2 : \delta^*(q_0, s = s'.i) = \delta(\delta^*(q_0, s'), i)$. Two cases: $i = 0 \vee i = 1$
   1. $i = 0$. $\delta(q_i, 0) = q_1 \forall q_i \in Q$
   2. $i = 1$. $\delta(\delta^*(q_0, s'), 1) = \delta(q_2, 1)$, since $s' \in L_3$. Then,$\delta(q_2, 1) = q_1$

   Thus, $\forall s \in L_2$, $\delta^*(q_0, s) = q_1$ is true
3. $\forall s \in L_3$: $\delta^*(q_0, s = s'.i) = \delta(\delta^*(q_0, s'), i) = \delta(\delta^*(q_0, s'), 1) = \delta(q_1, 1)$, since $s' \in L_2$. Then, $\delta(q_1, 1) = q_2$, thus $\forall s \in L_3$, $\delta^*(q_0, s) = q_2$ is true

Translators

- It is possible to enrich the computation model of a recogniser FSA with the ability to translate languages
- The FSA is only able to translate languages where the source is recognisable
- A translation move is characterised by reading at most one character and outputting zero or more characters
- A typical example is the find-replace function of a common editor

Formalization

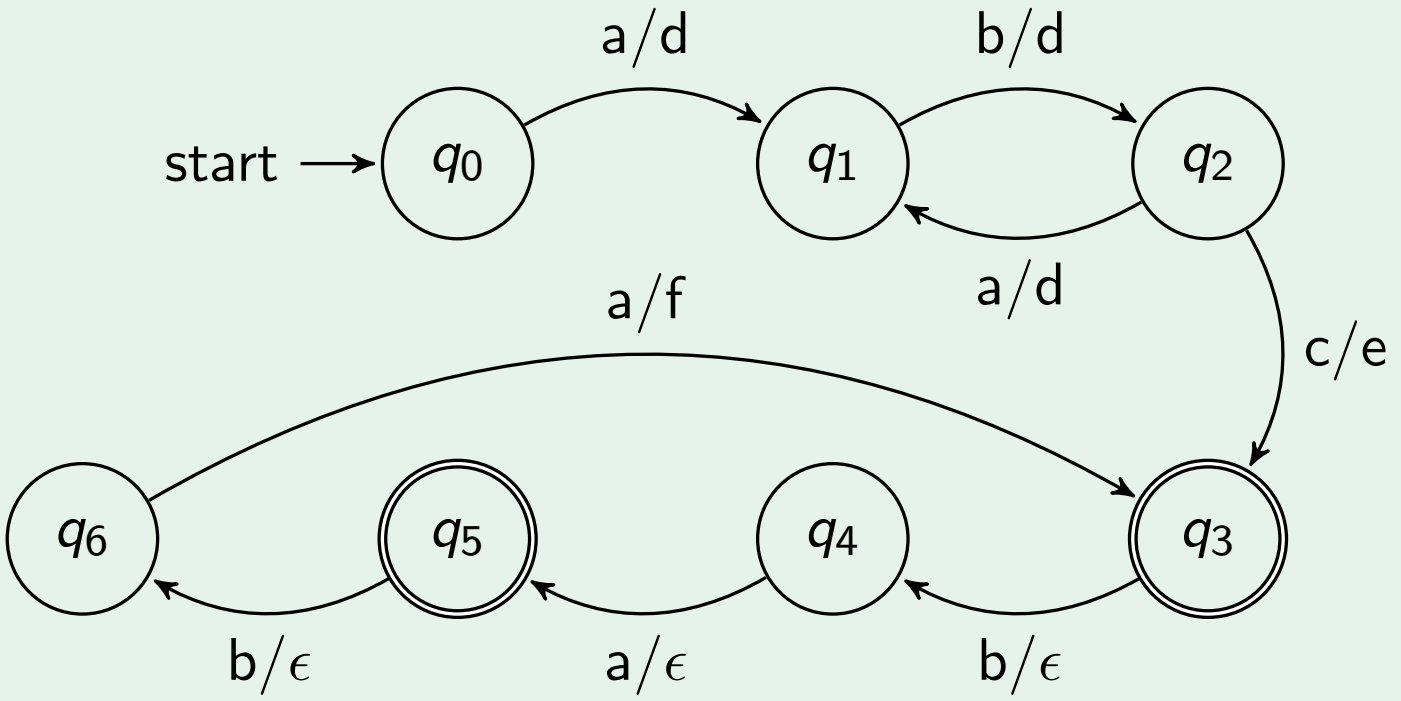
- A transducer FSA is formally defined as a 7-tuple $(\mathbf{Q}, \mathbf{I}, \delta, q_0, \mathbf{F}, \mathbf{O}, \eta)$, where:
  - $\mathbf{Q}$, is the set of states of the automata
  - $\mathbf{I}$ is the alphabet of the input string which will be checked
  - $\delta : \mathbf{Q} \times \mathbf{I} \mapsto \mathbf{Q}$ the transition function
  - $q_0 \in \mathbf{Q}$ the (unique) initial state from where the automaton starts
  - $\mathbf{F} \subseteq \mathbf{Q}$ the set of final accepting states of the automaton
  - $\mathbf{O}$ the output alphabet (may coincide with $\mathbf{I}$)
  - $\eta : \mathbf{Q} \times \mathbf{I} \mapsto \mathbf{O}^*$ the transduction function

Example

- We want to build a transducer automaton accepting the input language $L = (ab)^+c(ba)^*$
- The target language is $L_t = (dd)^+ef^*$
- The translation map τ is defined as $\tau((ab)^nc(ba)^m) = d^{2n}ef^{(m \div 2)}, n \geq 1, m \geq 0$
- Since only a finite memory is required to perform the $\div 2$ operation, the language can be translated by a FSA.

Transducer FSA

L to L_t transducer



Data Compression

We want to compress words from the language $L = \{a^{n_1} b^{k_1} \dots a^{n_i} b^{k_i} \dots a^{n_N} b^{k_N} c \mid N \geq 1 \wedge \forall i \leq N(1 \leq n_i \leq 4 \wedge 1 \leq k_i \leq 4)\}$ into a more compact, equivalently expressive format $L_t = n_1 k_1 \dots n_i k_i \dots n_N k_N \mid N \geq 1$.

L to L_t transducer