Languages and Finite State Automata Design

Alessandro Barenghi

Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

alessandro -dot- barenghi -at- polimi -dot- it

March 4, 2021

Languages

Quick recap

- An alphabet is a *finite* set of *symbols*
- A language is a set of *sequences* of symbols from an alphabet
  - Sequences go by the names: *word*, *string*, *phrase*
  - A language is not necessarily a finite set
  - The empty word ϵ may be part of a language
- Sometimes in this course we will specify a language in natural language: effective for our purposes, but rather informal
  - Example: given an alphabet $\mathbf{I} = \{a, b\}$, consider the language L of all strings with an even number of a

Languages

Notation (examples on $\mathbf{I} = \{a, b\}$)

- The concatenation operator is often omitted: $a.b$ becomes ab
- It is defined on letters, naturally extended on words
- Iterated concatenation is written as exponentiation: $aaaa$ becomes a^4 (mimics multiplication)
  - But concat does not commute: $a^2b^2 = aabb \neq abab = (ab)^2$
- Languages are sets, we write them enclosed in curly braces:
  - Enumerating them: $L = \{ab, aaba\}$, $L = \{\epsilon\}$
  - In natural language:
    $L = \{s, \text{where } s \text{ is a string with an odd number of } a\}$
  - Something in between $L = \{a^n, n \in \mathbb{N} \text{ and } n > 3\}$
- N.B. $abab$ is a word, $\{abab\}$ is a set of words (language)

Set operations

Union $\cup$, Intersection $\cap$ and difference $\setminus$

- What's $\{a^4\} \cup \{a^{2n}, n \in \mathbb{N}\}$? $\{a^{2n}, n \in \mathbb{N}\}$
- What's $\{a^{2n+1} n \in \mathbb{N}\} \cup \{a^{2n}, n \in \mathbb{N}\}$? $\{a^n, n \in \mathbb{N}\}$
- What's $\{a^6\} \cap \{a^{2n}, n \in \mathbb{N}\}$? $\{a^6\}$
- What's $\{b\} \cap \{a^{2n}, n \in \mathbb{N}\}$? $\{\}$
- What's $\{a^n, n \in \mathbb{N}\} \cap \{b^n, n \in \mathbb{N}\}$? $\{\epsilon\}$
- What's $\{a^4\} \setminus \{a^{2n}, n \in \mathbb{N}\}$? $\{\}$
- What's $\{a^{2n}, n \in \mathbb{N}\} \setminus \{a^4\}$? $\{a^{2n}, n \in \mathbb{N} \setminus \{2\}\}$

Language concatenation

From an operative standpoint

- $L_1.L_2 =$ the language of all the strings obtained concatenating a string from L_1 and a string from L_2 *in this order*

Warm-up: $L_1 = \{(ab)^n, n \in \mathbb{N}\}$, $L_2 = \{a^n b^n, n \in \mathbb{N}\}$

- What is $L_1.L_2$?
- What is $(L_1.L_2) \cap L_1$? L_1
- What is $(L_1.(L_2 \setminus \{\epsilon\})) \cap L_1$? $L_1 \setminus \{\epsilon\} = \{(ab)^n, n \in \mathbb{N} \setminus \{0\}\}$
- What is $L_1.L_1$? L_1
- What is $L_2.L_2$? $\{a^n b^n a^m b^m, n \in \mathbb{N}, m \in \mathbb{N}\}$

Looking at the stars

On an alphabet

- Applied to a set of *letters*, yields an (infinite) set of *words*
  - It generates the set of all the words writeable with the alphabet
- N.B. $\mathbf{I} = \{a\}, \mathbf{I}^* = \{a^n, n \in \mathbb{N}\}$, $\mathbf{I}^*$ is a *language*
- We denote $\mathbf{I}^* \setminus \{\epsilon\}$ as $\mathbf{I}^+$

On a language

- L^* is the union of all the elements of $L^n, n \in \mathbb{N}$
- What is L^* with $L = \{\epsilon, aa, ab, ba, bb\}$? $\{a, b\}^* \setminus \{a, b\}$

Complement

- The complement $\bar{L}$ (also $\neg L$) of L over the alphabet $\mathbf{I}$ is $\mathbf{I}^* \setminus L$

Regular expressions

A compact way to describe languages

- A regular expression $\mathtt{R}$ denotes a language $L(\mathtt{R})$:
  - a constant:$\varnothing$ (empty language), ϵ, or a letter in $\mathbf{I}$
  - $\mathtt{R}.\mathtt{S}$: $L(\mathtt{R}).L(\mathtt{R})$
  - $\mathtt{R}|\mathtt{S}$: $L(\mathtt{R}) \cup L(\mathtt{R})$
  - (round) parentheses enforce precedence
  - $\mathtt{R}^*$: $L(\mathtt{R})^*$
  - $\mathtt{R}^+$: $L(\mathtt{R})^+$
- As expressive as finite state automata (we will not prove it... in this course): can algorithmically be transformed in them
- Really handy to describe in a compact fashion a language

Finite State Automata

What are FSAs?

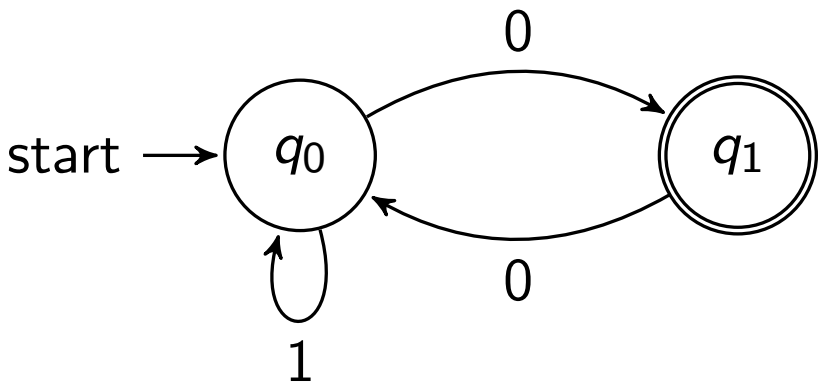
- Finite State Automata (Automi a Stati Finiti ASF) are the simplest among abstract computing models we'll see
- They are still able to describe a good deal of useful systems
- They are characterized by a *finite memory capacity*
- In a sense, every real world computing machine is a FSA (as it only has a finite number of memory elements)
- However modeling it as such is not a good idea (too many states, roughly $2^{2^{43}} \simeq 10^{300000000000}$ for a common PC)

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

A first recognizer

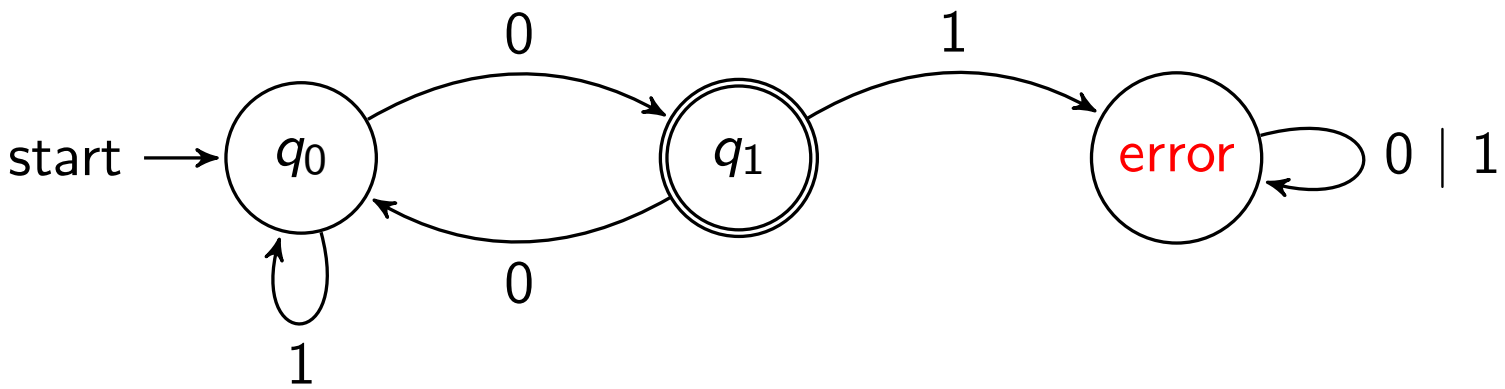
- Let's analyze a recognizer to understand the language it recognizes



- Recognizes $(1|00)^*0$
- $\delta(q_1, 1)$ 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

Complement

How-To

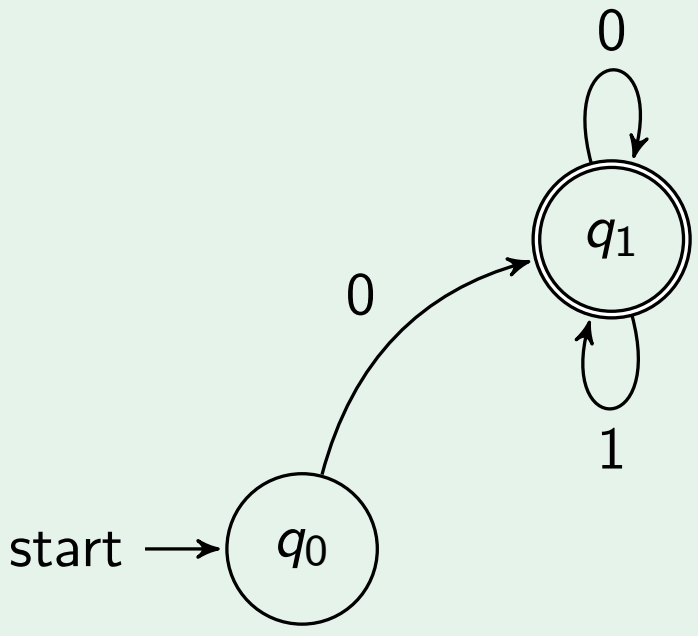
- Given a recognizer FSA for the language L, the one recognizing the complement of the language $\bar{L}$ can be obtained flipping the termination condition
- Basically: all the accepting states become non accepting and vice-versa
- Take care: the error state is nothing more than a common non accepting 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

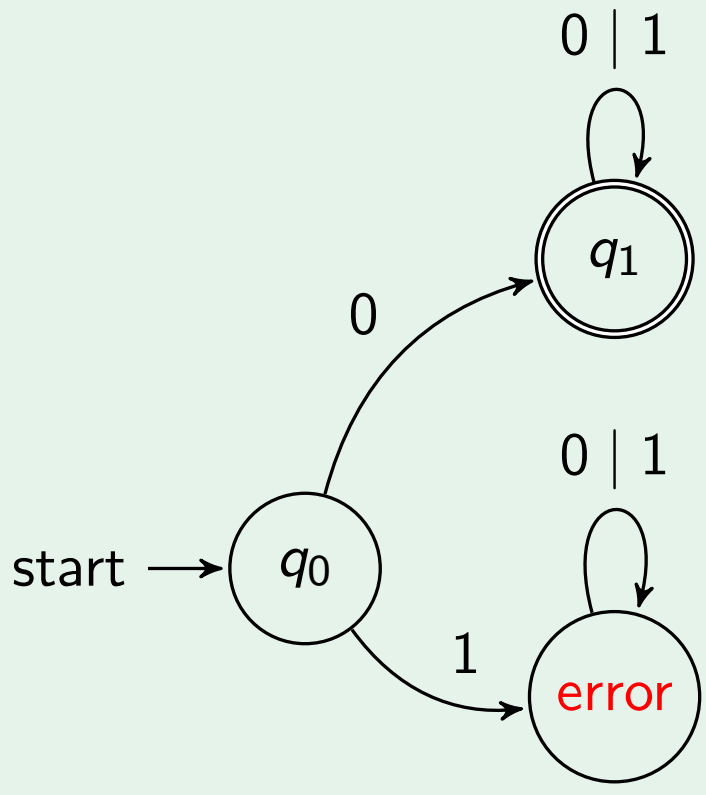
- Let's take as an example the recognizer for $L = 0(0|1)^*$

The recognizer looks like this :

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



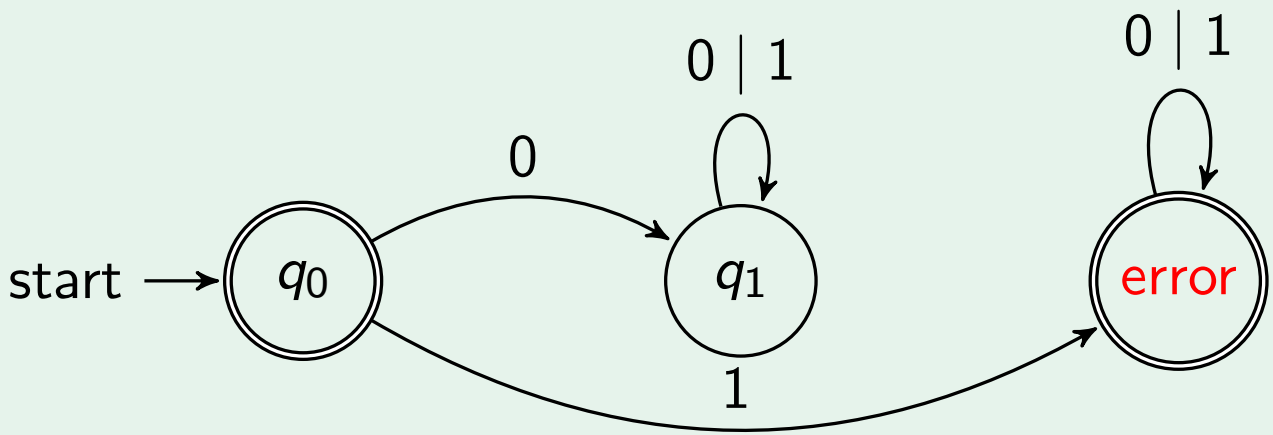
Complete $\delta(\cdot, \cdot)$



Complement

- The recogniser for $\bar{L}$ is thus :

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



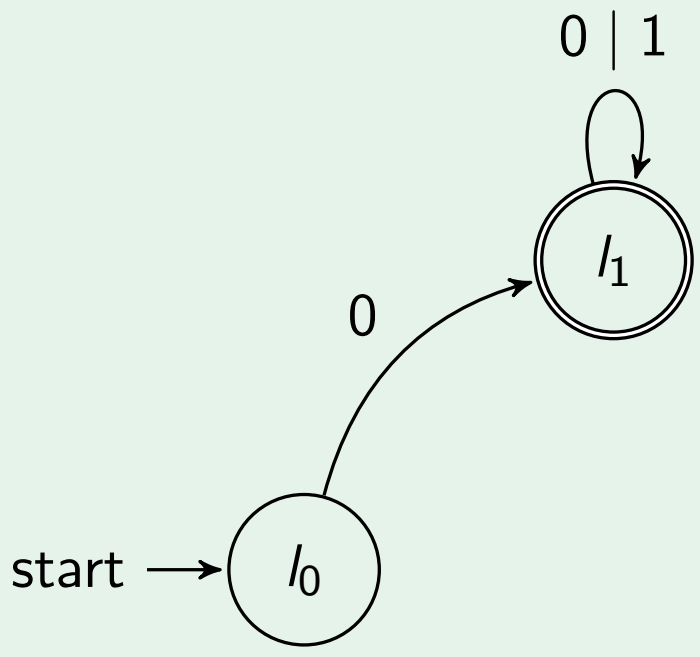
Intersection

- 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 $l_i \in F_l \wedge m_i \in F_m$

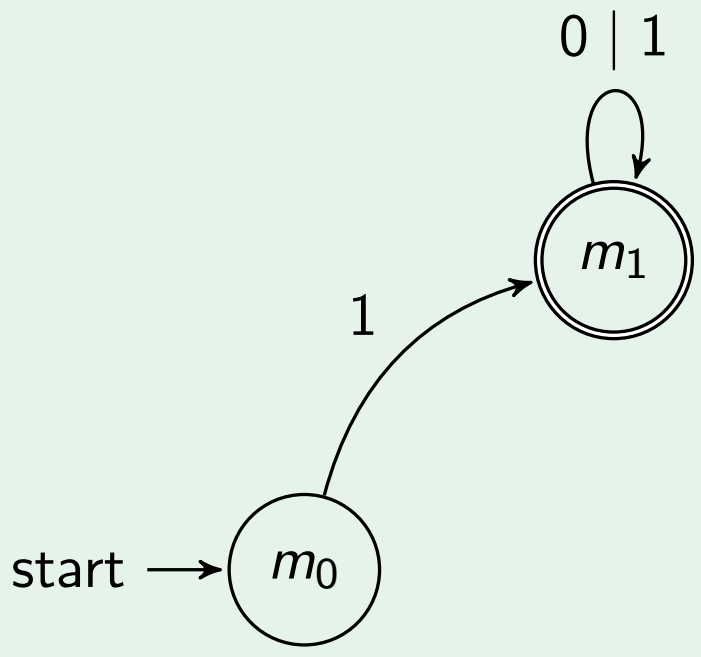
Example intersection

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

Recogniser for L

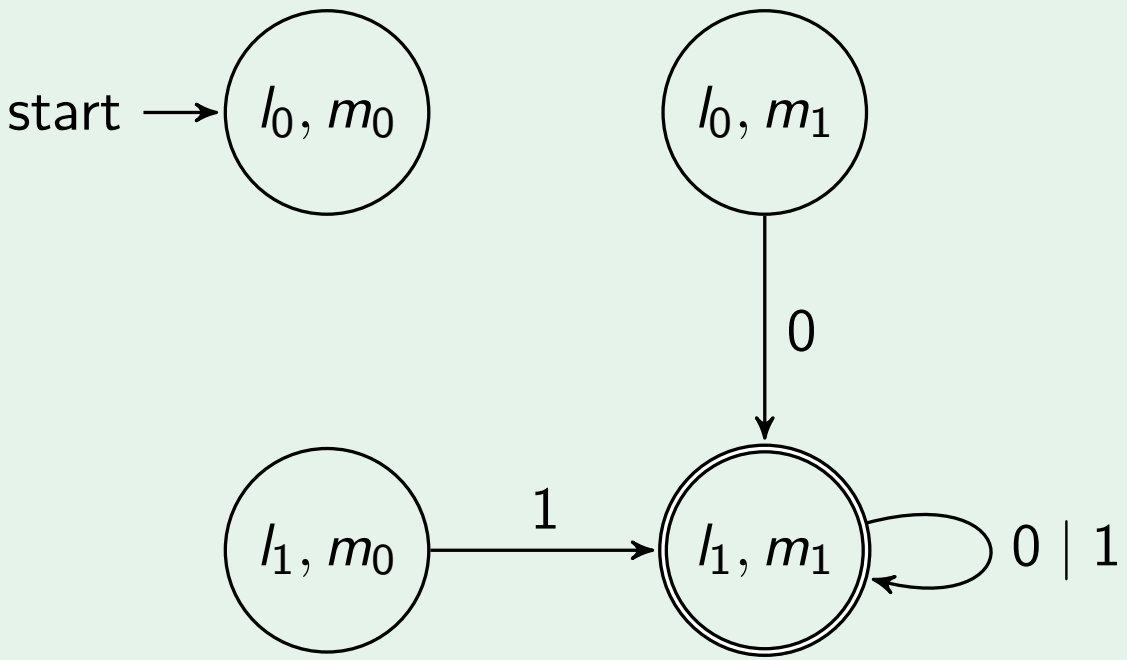


Recogniser for M



Intersection Automaton

Resulting automaton (including unreachable states)



Union

- 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 , applying in sequence two complements an intersection and another complement we obtain the union automaton

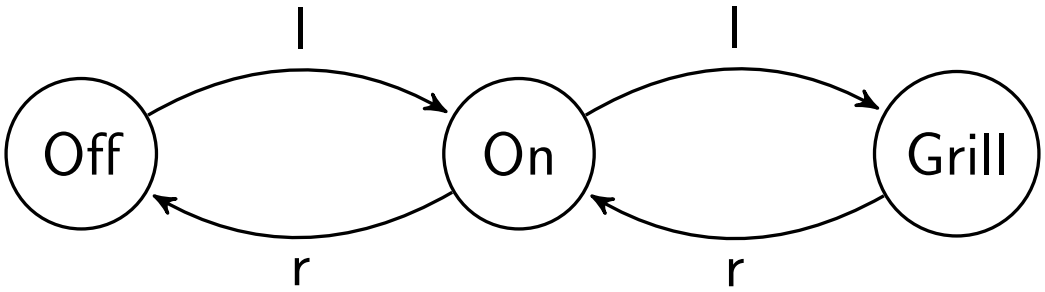
Extra Material

Modeling a real object

An oven knob

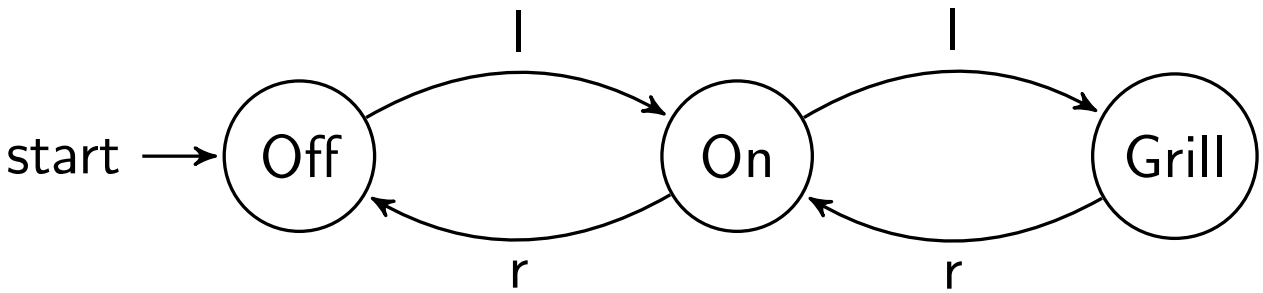
- We will start looking at the design for a simple automaton representing an oven knob
- The oven may be *On*, *Off* or on *Grill* position.
- The knob turns left and right: left raises the temperature, right lowers it.
- Our FSA will recognize only the sequences of turns defining a "good" operating sequence

A first attempt



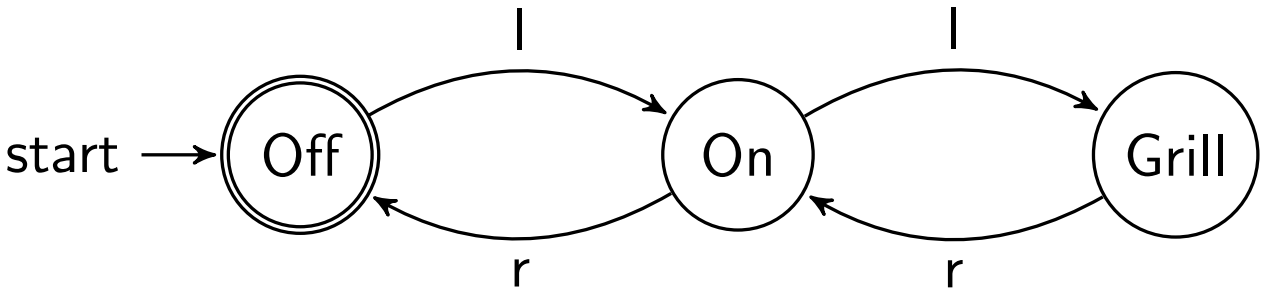
But this oven starts in an undefined state...

A first attempt



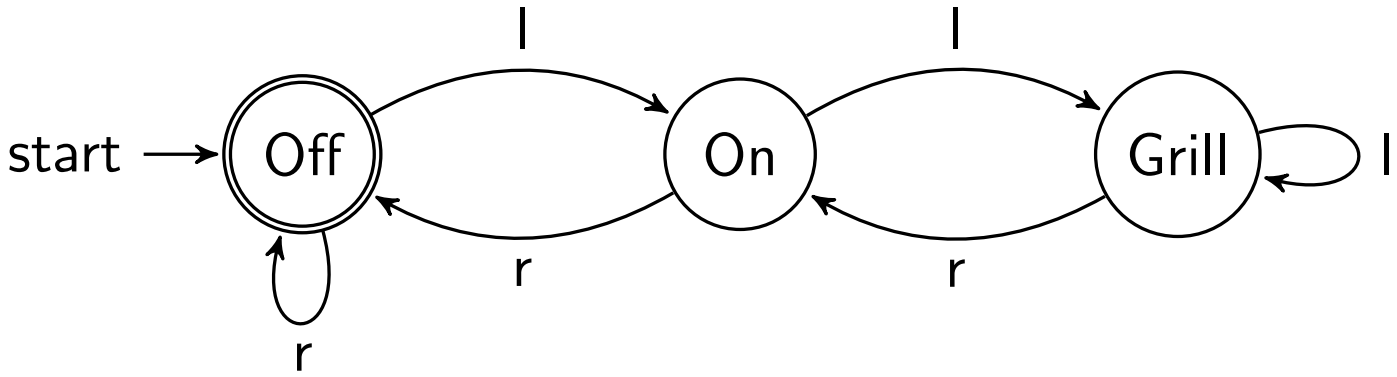
This one's better, but we'd like to accept only when we remembered to turn it off at the end of the cooking...

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? $L = (r \mid I(I(I)^* r)^* r)^* = (r \mid I(I^+ r)^* r)^*$