Models of Computation: Limitations of the Regular Languages

The Pumping Lemma

Dr Kamal Bentahar

School of Engineering, Environment and Computing
Coventry University

24/10/2016

Regular Languages

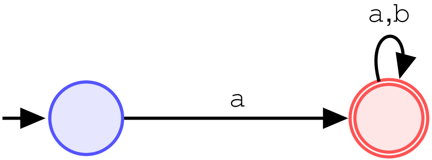
Regular Languages

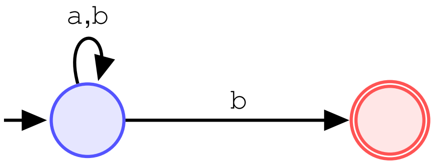
The class of regular languages can be:

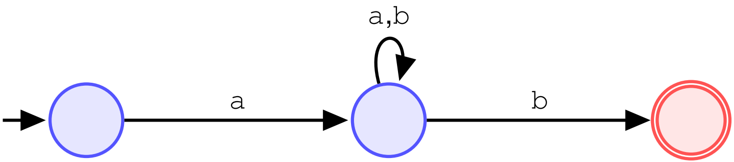
1. Recognized by NFAs. (equiv. GNFA or ε-NFA or NFA or DFA).
2. Described using **Regular Expressions**.
3. Generated using **Regular Grammars**. (See this later. . .)

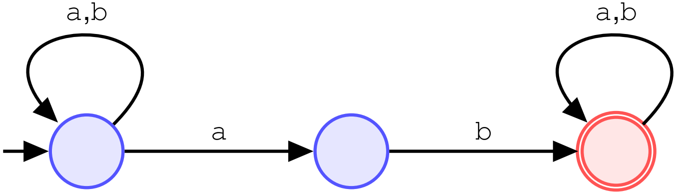
A look back

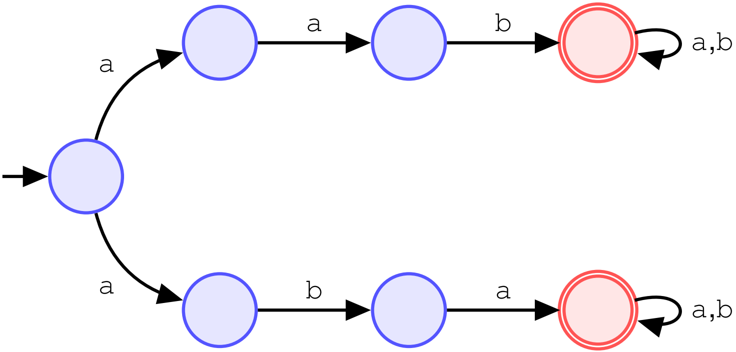
Let us look back at some examples. . . for each automaton in the next slides, let us think about **equivalence of NFA/DFA**, **RegEx**, and the **path taken by an accepted string** (is it “straight” or does it loop?).

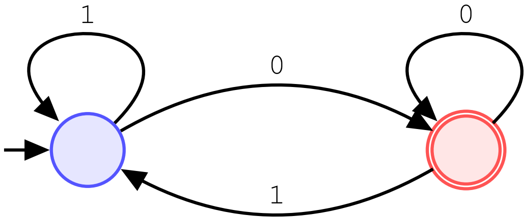

a
a,b

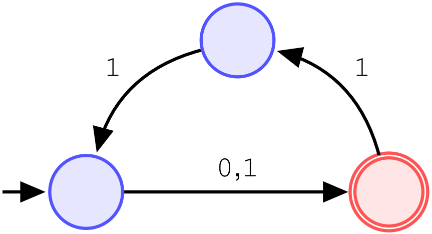

a,b
b

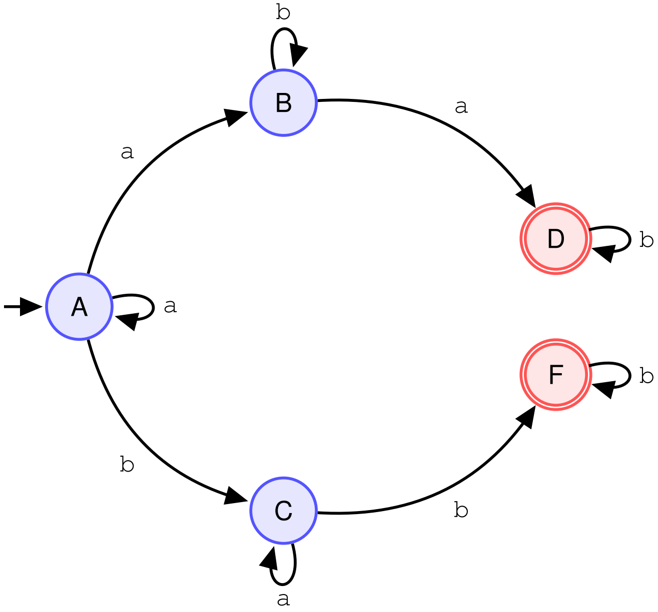

a,b
a
b


a,b
a
b
a,b

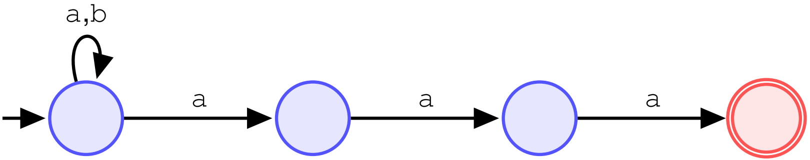

a
a
b
a,b
a
b
a
a,b

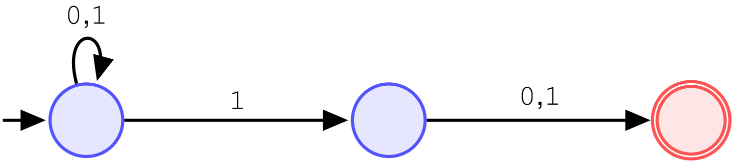

1
0
0
1

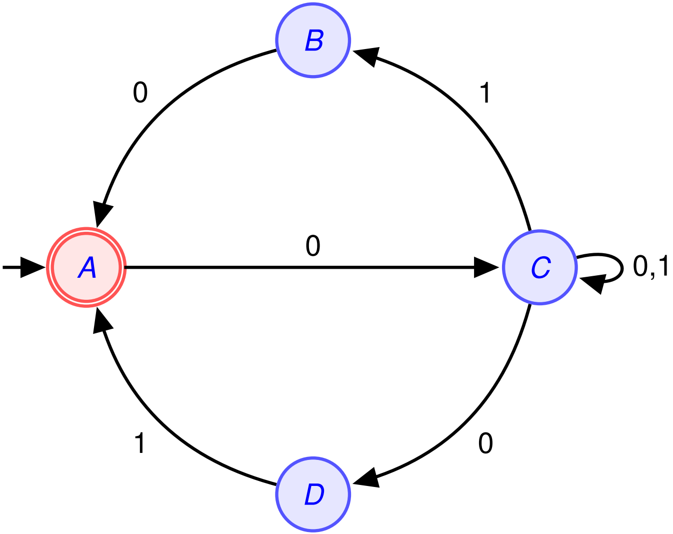

1
1
0,1

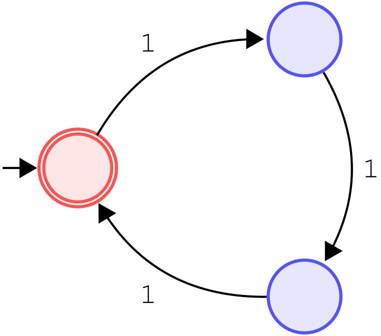


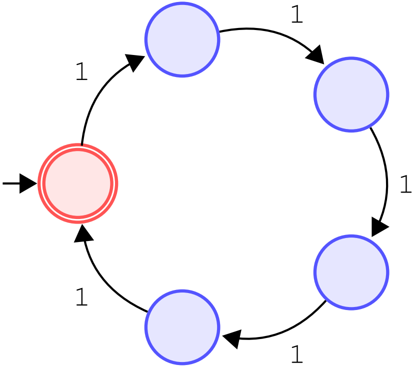
A
B
C
D
F
a
a
b
b
a
a
b
b
b

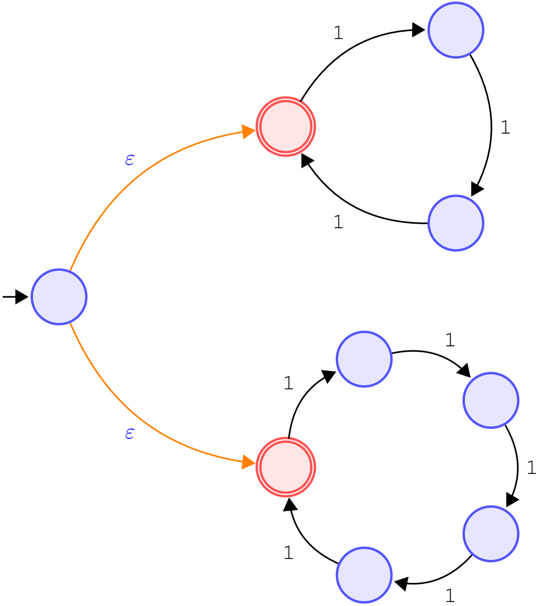

a,b
a
a
a


0,1
1
0,1


B
0
1
A
0
C
0,1
1
0
D


1
1
1


1
1
1
1
1



1
1
1
ε
ε
1
1
1
1
1

Special case

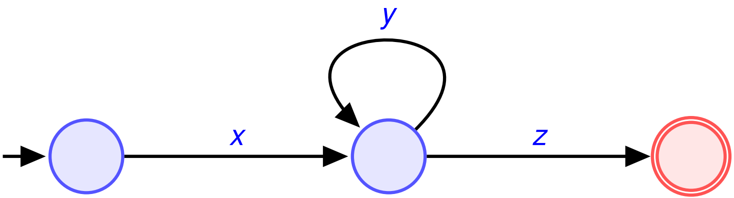
Let us investigate languages over unary alphabets, e.g. $\Sigma = \{a\}$.
The language is

- either **finite**, in which case it is regular trivially
- or **infinite**, in which case its DFA will have to **loop**.

Pigeon-hole principle: if you put more than n pigeons into n holes then there must be a hole with more than one pigeon in.

The Pumping Lemma

Observation: path from the start to the accept state for a string xyz:



The strings x and y can be ε.

Idea of the Pumping Lemma

Any "sufficiently long" string in a regular language can be broken into three parts such that if we **"pump" the middle part** (repeat it zero or more times) then the result would still be in the language.

Regular Languages (RLs) – proofs

We have learnt how to show that a language is regular by

- constructing a DFA/NFA recognizing it,
- or by writing a Regular Expression for it.
  Exploiting closure under union, concatenation and star.

(Proof by “existence”)

However, *not all languages are regular*; so how can we show that a given languages is **not** regular?!

Proof by “contradiction”

DFAs have a finite number of states, so once a string gets beyond a certain length, the DFA must repeat one or more states.

- When a DFA repeats a state (say q_8), we may divide the input string up into three substrings:
  1. The substring x before the first occurrence of q_8
  2. The substring y between the first and last occurrence of q_8
  3. The substring z after the last occurrence of q_8
- It follows that if the DFA accepts xyz, then it will also accept $xy, xyyz, xyyyz, \ldots$

Therefore, for any RL, once a string extends above a certain length (the pumping length p) it becomes possible to divide the string up into three substrings xyz, in such a way that xy^*z is also a member of that language

- x, z can be ε
- y cannot be ε
- $|xy| \leq p$

Pumping Lemma

Pumping Lemma

Let L be a regular language. Then there exists a constant p such that for every string w in L, with $|w| \geq p$, we can break w into three strings $w = xyz$ such that

1. $y \neq \varepsilon$ (or equivalently $|y| > 0$)
2. $|xy| \leq p$
3. For all $k \geq 0$, the string xy^kz is also in L

The length p is called **the pumping length**.

Its main purpose in practice is to prove that a language is not regular.

That is, if we can show that a language does not have the required property, then we can conclude that it cannot be expressed as a regular expression or recognized by a DFA.

The Pumping Lemma when used to prove that a language L is **not regular** can be viewed as a "game" between a **Prover** and a **Falsifier** as follows:

❶ **Prover** claims L is regular and fixes the pumping length p.

❷ **Falsifier** challenges **Prover** and picks a string $w \in L$ of length at least p symbols.

❸ **Prover** writes $w = xyz$ where $|xy| \leq p$ and $y \neq \varepsilon$.

❹ **Falsifier** wins by finding a value for k such that xy^kz is **not** in L. If it cannot then it fails and **Prover** wins.

The language L is not regular if **Falsifier** can always win systematically.

Example ($L = \{\mathtt{a}^n\mathtt{b}^n \mid n \geq 0\}$)

❶ **Prover** claims L is regular and fixes the pumping length p.

❷ **Falsifier** challenges **Prover** and picks $w = \mathtt{a}^p\mathtt{b}^p \in L \quad (|w| = 2p \geq p)$.

❸ **Prover** tries to write w as $w = xyz$ but sees that the condition $|xy| \leq p$ forces x and y to only contain the symbol $\mathtt{a}$. Also, y cannot just be the empty string because of the condition $y \neq \varepsilon$. So the only option available is to have $xy = \mathtt{a}^m$ for some $m \geq 1$, and then we get $z = \mathtt{a}^{p-m}\mathtt{b}^p$.

❹ **Falsifier** now sees that $xy^0z, xy^2z, xy^3z, \ldots$ all do not belong to L because they either have less or more $\mathtt{a}$'s than there are $\mathtt{b}$'s. So, any such string will be enough for **Falsifier** to win the game.

Example ($L = \{ww \mid w \in \{0,1\}^*\}$)

❶ **Prover** claims L is regular and fixes the pumping length p.

❷ **Falsifier** challenges **Prover** and Choose $w = (0^p1)(0^p1) \in L$. This has length $|w| = (p+1) + (p+1) = 2p + 2 \geq p$.

❸ **Prover** The PL now guarantees that w can be split into three substrings $w = xyz$ satisfying $|xy| \leq p$ and $y \neq \varepsilon$.

❹ **Falsifier** Since $w = (0^p1)(0^p1) = xyz$ with $|xy| \leq p$ then we must have that y only contains the symbol 0. We can then pump y and produce $xy^2z = xyyz \notin L$, causing a contradiction.
So L is not regular.

Food for thought

- If modern computer = finite state machine: a finite amount of data, say $1\text{TB} = 1024^4 \times 8 = 2^{43}$ bits of information, i.e. a maximum of $2^{2^{43}} \approx 10^{2,647,887,844,335} \approx 10^{10^{12.4}}$ states – a finite number still!
- Consequently, my computer is unable to recognize the language $a^n b^n$
- This means that at some point, my computer can no longer count the number of a's in a string... This occurs when the number of a's becomes greater than $2^{2^{43}}$.
- We are assuming that the computer is not storing the string (in which case it would just run out of memory anyway)
- At 3GHz, this would take... a length of time so inconceivably huge that the age of the universe would be negligible by comparison

Space Complexity: Constant Space $\Longleftrightarrow$ NFAs

- Finite State Automaton: good model for algorithms which require **constant space**.
  Space complexity $O(1)$, i.e. space used does not grow with respect to the input size.
- Some languages cannot be recognized by FSMs.
  Space used must grow with respect to input size.
- We will see a more powerful model of computation next week :-)