

CS 3313

Foundations of Computing:

Properties of Regular Languages: Non-regular Languages

<http://gw-cs3313-2021.github.io>

Properties of Regular Languages

- Closure Properties: what happens when we “combine” two regular languages or perform set operations on them ?
 - Ex: Is Intersection of two regular languages still a regular language ?
 - Why is this important ?
 - Construct a larger set from smaller sets
 - Problem decomposition
- Decision Problems: can we provide procedures to determine properties of a language ?
 - Ex: are two machines equivalent? Does a DFA accept an infinite set ?
- How do we determine if a language does not belong to that class of languages ?
 - Ex: How do we show that a language (problem?) cannot be accepted by a DFA ?

Frequently used concept: Product DFA

- “compose” two DFAs using cartesian product of their states
- Let M_1 and M_2 be two DFAs with states Q and R
 - $M_1 = (Q, \Sigma, \delta_1, q_0, F_1)$ and $M_2 = (R, \Sigma, \delta_2, r_0, F_2)$
- Product DFA M_p :
- Product DFA has set of states $Q \times R$
 - i.e., pairs $[q, r]$ with q in Q and r in R
- Start state = $[q_0, r_0]$ (the start states of the two DFA's).
- **Transitions:** $\delta([q, r], a) = [\delta_1(q, a), \delta_2(r, a)]$
 - δ_1, δ_2 are the transition functions for the DFA's of M_1, M_2
 - That is, ***we simulate the two DFA's in the two state components of the product DFA.***
- Note: we have not yet defined the final states of the product DFA

Summary of Closure Properties

- Regular languages are closed under Union, Concatenation, star closure, complementation, reversal, intersection, homomorphism (and reverse homomorphisms)
- Where are closure properties used ?
 - Construction a solution (DFA or Reg. Expr.) for a larger language using simpler solutions (machines or languages)
 - Analogy: modular composition of software modules
 - Useful in simplifying proofs to show a language is not regular
 - Useful in constructing “decision algorithms”

Decision Properties of Regular Languages

- A *decision property* for a class of languages is an algorithm that takes a formal description of a language (e.g., a DFA) and tells whether or not some property holds.
- We say that the property is *decidable* if there is an algorithm that determines if the property holds

Decision Properties for Regular Languages

- Membership: Is w in $L(M)$?
- Emptiness: Is $L(M)$ empty ?
- Equivalence: Is $L(M1) = L(M2)$?
- Subset: Is $L(M1)$ a subset of $L(M2)$?
- Infiniteness: Is $L(M)$ infinite ?

The Infiniteness Problem

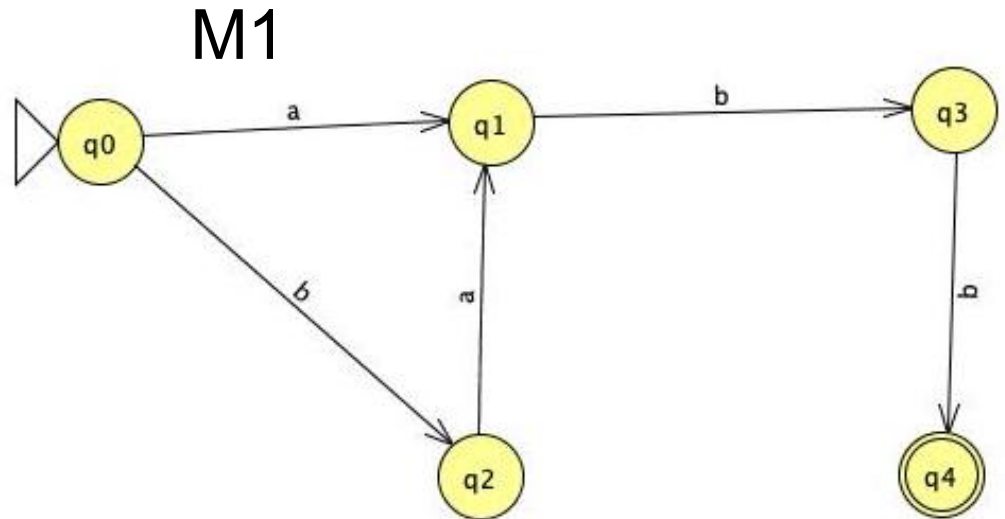
- Is a given regular language infinite?
- Theorem: Testing if $L(M)$ is infinite is a decidable problem.
- Start with a DFA for the language.
- **Key idea:** if the DFA has n states, and the language contains any string of length n or more, then the language is infinite.
- Otherwise, the language is surely finite.
 - Limited to strings of length n or less.

$L(M)$ infinite ?

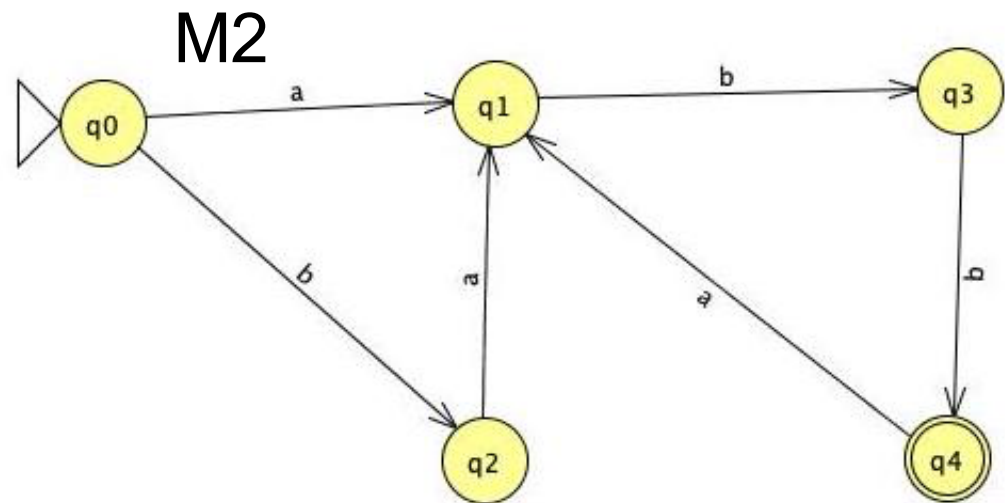
- Proof: use the graph representation to present the procedure/proof.

Transition graphs for two DFAs

Is $L(M1)$ finite ?



Is $L(M2)$ finite ?



Algorithm to test for $L(M)$ infinite

- Input: Transition graph for DFA M
- Output: Yes if $L(M)$ is infinite, No if $L(M)$ is finite
- Algorithm ?
- Check if graph has a cycle!

So what kinds of languages are not regular and how do we prove they are not ?

- Proof for testing infiniteness of $L(M)$ reveals some properties that can be used to prove that a language is not regular.
- Given any language L , it is either regular or it is not.
 - To prove L is regular, we have to provide a DFA/NFA or Regular expression that accepts L .
 - To prove L is not regular, we need to provide a formal proof using some properties of all regular languages
 - Simply saying “I spent a lot of time and could not find a DFA” is NOT a proof.

Why is it useful to ask if a language is Regular (or Context free or ...)

- Example: Can we check if there are syntax errors in a C program by using a DFA ?
 - Syntax checking is first step in a compiler's translation process
 - Program must satisfy the rules (specified as a grammar) of the C programming language (or any programming language)

Power of abstraction

- If a DFA can do syntax checking, then a DFA can check if there are an equal number of left and right braces ({ and } are used to specify a code block in C)
 - Choose $L = \{ w \mid w \text{ is a string over } a,b \text{ and } w \text{ has equal number of } a\text{'s and } b\text{'s} \}$
 - Using a to denote { and b to denote } (recall homomorphism which will let you substitute symbols)
- Now apply closure properties: we know that a^*b^* is regular and regular languages are closed under intersection
 - Therefore $L_1 = L \cap a^*b^* = \{ a^i b^i \mid i \geq 1 \}$ must be a regular language
 - Equal number of a 's and b 's
- So, is L_1 a regular language ?

“power” of DFAs: A little intuition

- So what can DFAs (i.e., finite state machines) “compute” ?
- What can they store and where ?
 - State
- Do they have an “external” memory to store a value ?

Paths in a graph

- Graph has n vertices
- Path in a graph can be represented as a sequence of vertices: $v_1v_2v_3\dots v_k$ where (v_i, v_j) is an edge
- Suppose we have a path of length n , how many vertices on the path ?

DFA Transition Graph

- The transition function of a DFA can be represented as a (directed graph).
- DFA has a finite number of states: n
- Suppose there is a string of length $\geq n$ accepted by the DFA
 - Vertex sequence in the path = ?

Cycles in the path

The Pumping Lemma for Regular Languages

For every regular language L

*Number of
states of
DFA for L*

There is an integer n , such that

For every string w in L of length $\geq n$

We can write $w = xyz$ such that:

1. $|xy| \leq n$.
2. $|y| > 0$.
3. For all $i \geq 0$, xy^iz is in L .

*Labels along
first cycle on
path labeled w*

Example: $L = \{ a^i b^i \mid i \geq 0 \}$

How do use the pumping lemma: 2 person adversarial game

- **For all** regular languages L , **there exists** n ...**for all** w in L ...**there exists** xyz
- Logical statements/assertions that have several alternations of for all and there exists quantifiers can be thought of as a game between two players
- Application of the pumping lemma can be seen as a two player game (of 5 steps)
- Example: $L = \{ ww \mid w \text{ in } \{a,b\}^* \}$

Pumping Lemma as Adversarial Game

- 1: Player 1 (me) picks the language to be proved nonregular
- 2. Player 2 picks n , but doesn't reveal to player 1 what n is; player 1 must devise a play for all possible n 's
- 3. Player 1 picks w , which may depend on n and which must be of length at least n

Pumping Lemma as Adversarial Game

- 4: Player 2 divides w into x,y,z obeying the constraints that are stipulated in the lemma: y is not empty and $|xy| \leq n$.
 - Again, Player 2 does not tell Player 1 what xyz are; just that they obey the constraints
- 5. Player 1 “wins” by picking k , which may be a function of n, x, y , and z such that xy^kz is not in L .

Power of abstraction & Combining theorems

- If a DFA can do syntax checking, then a DFA can check if there are an equal number of left and right braces ({ and } are used to specify a code block in C)
 - Choose $L = \{ w \mid w \text{ is a string over } a,b \text{ and } w \text{ has equal number of } a\text{'s and } b\text{'s} \}$
 - Using a to denote { and b to denote } (recall homomorphism which will let you substitute symbols)
- Now apply closure properties: we know that a^*b^* is regular and regular languages are closed under intersection
 - Therefore $L_1 = L \cap a^*b^* = \{ a^i b^i \mid i \geq 1 \}$ must be a regular language
 - Equal number of a 's and b 's
- So, is L_1 a regular language ?

Exercise:

- $L = \{ ww^R \mid w \text{ in } \{a,b\}^* \}$

More examples...

More examples...

More examples...

More examples...