CS304: Automata and Formal Languages

Lec 12

Equivalence of States and DFAs and DFA Minimization

Rachit Nimavat

Aug 29, 2025

Outline

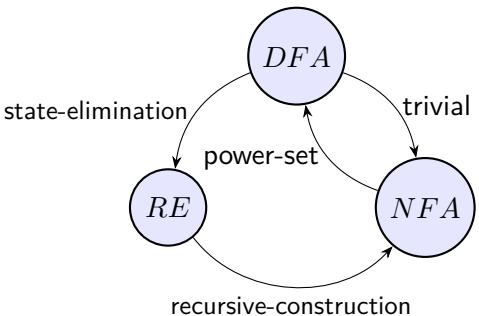
Regular Language

Defn: A language L is called a regular language if and only if there exists a regular expression R that describes it: $L = L(R)$

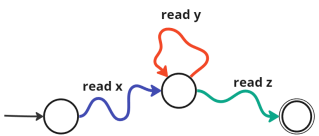
Theorem 1.1 (Kleene's Theorem)

All these definitions are equivalent:

- $\exists$ *a DFA* D *such that* $L = L(D)$
- $\exists$ *an NFA* A *such that* $L = L(A)$
- $\exists$ *an RE* R *such that* $L = L(R)$



Pumping Lemma



Lemma 1.2

For each regular language L, there is a constant p (the pumping length) such that any string $s \in L$ with $|s| \geq p$ can be divided into three parts, $s = xyz$, such that

i. $|y| > 0$ *("looped-string" y is not empty)*

ii. $|xy| \leq p$ *(the loop starts within the first p characters)*

iii. $\forall i \geq 0$, *the string* $xy^iz \in L$ *(we can pump the loop zero, one, or many times, and the resulting string must still be accepted)*

How to use Pumping Lemma?

Goal: Prove that a language L is irregular.
Assume that L is regular.

Pumping Lemma: Player-1

1. Provides 'pumping length' p

3. Provides a partition $s = xyz$ such that $|y| > 0$ and $|xy| < p$

Us: Player-2

2. Cleverly choose $s \in L$ such that $|s| \geq p$

4. WIN by choosing $i \geq 0$ such that $xy^iz \notin L$

If $xy^iz \notin L$, then we have a contradiction. Thus, our assumption that L is regular must be false. Therefore, L is irregular.

Note: For this to work, we must have a winning strategy for each p and for each partition $s = xyz$ (that satisfies $|y| > 0$ and $|xy| < p$).

Example

Prove that $L = \{0^m 1^n 2^n \mid m, n \geq 0\}$ is not regular

Assume for contradiction that L is regular

From pumping lemma, there is a constant p with "looping property"

Let's pick a string $s = 0^p 1^p 2^p$. Notice that $s \in L$

What if pumping lemma plays $s = (\varepsilon)(0^p)(1^p 2^p)$? We do not have winning strategy here!

Let's pick a string $s = 1^p 2^p$. Notice that $s \in L$

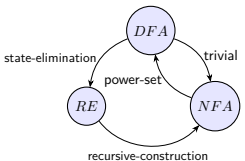
Consider its decomposition $s = xyz$ given by the pumping lemma

$|xy| \leq p \implies y = 1^{|y|}$ y consists of all ones

Choose $i = 0$. From pumping lemma, $s_0 = xy^0 z = xz = 1^{p-|y|} 2^p$. But then $s_0 \notin L$.

Contradiction! since $|y| > 0$)

Closure Properties



Closure Properties: A set is closed under an operation if applying that operation to its elements results in elements that are also in the set

- Regular languages are closed under union Regular expressions!
- Regular languages are closed under complementation DFAs!
- Regular languages are closed under intersection De Morgan's Laws!
- Difference of regular languages is regular $A \setminus B = A \cap \overline{B}$
- Reversal of regular languages is regular Reverse DFA and add a new start state with ε-transitions to old accept states
- Kleene-star of regular languages is regular Regular expressions!
- Concatenation of regular languages is regular Regular expressions!
- Substitution of characters by strings (**Homomorphism**) in regular languages is regular Regular expressions!
- Inverse Homomorphism is also regular, but don't worry about that. See Chap 4.2 of ALC for details.

Decision Properties

Three major questions:

1. Is the given regular language L empty?
   L is empty iff the DFA has no **reachable** accept states.
2. Is the given regular language L finite?
   L is infinite iff NFA has a 'loop' on a path to accept state. You'll learn an algorithm for this task in the Data Structures and Algorithms course.
3. Is the given regular language L equal to another regular language K?
   Check whether $(L \backslash K) \cup (K \backslash L)$ is empty or not. Is there a more direct way? What if each regular language has one single, standard, best DFA? Then we can just convert both L and K to this form and check if they are the same. For this, need to define **equivalence of DFA states**.

Indistinguishable States

Consider a DFA $M = (Q, \Sigma, \delta, q_0, F)$.

Definition 3.1

Two states, p and q, are indistinguishable (or equivalent) if for every possible string w, you end up in an accepting state starting from p if and only if you end up in an accepting state starting from q:

$$\hat{\delta}(p, w) \in F \iff \hat{\delta}(q, w) \in F$$

No Global View

Think from the perspective of the computer processing a string w on M

It doesn't need to load the entire M in memory

From any given state, the only thing it knows is which transitions are available

The Algorithm: How to Find Indistinguishable States?

Base Case: Mark any pair (p, q) as distinguishable if one is an accepting state and the other is not.

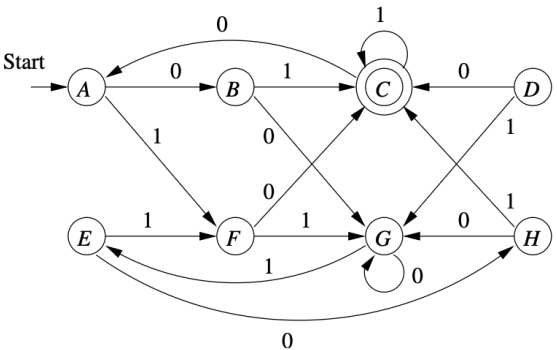
Inductive Step: For each pair of states (p, q), if there exists a symbol $a \in \Sigma$ such that $\delta(p, a)$ and $\delta(q, a)$ are known to be distinguishable, then mark (p, q) as distinguishable.

Repeat: Until no new pairs are marked in a full pass

Conclusion: Any pair of states that has not been marked as distinguishable is indistinguishable

HW: Prove that this algorithm terminates and its correctness. See, Chapter 4.4 in ALC.

Example



	A	B	C	D	E	F	G
B	x						
C	x	x					
D	x	x	x				
E		x	x	x			
F	x	x	x		x		
G	x	x	x	x	x	x	
H	x		x	x	x	x	x

See, Chapter 4.4.1 in ALC.

How to check if two Regular Languages are Equal?

Convert both languages to their respective DFAs

Check if the two DFAs are equivalent

How do we check if two DFAs are equivalent?

Consider two DFAs $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$
Check if states q_{01} and q_{02} are indistinguishable in $M = M_1 \cup M_2$

Does it matter that M has 2 start states?

NO! For indistinguishability, the start state is irrelevant

Observation 1 (Equality of Regular Languages)

$L(M_1) = L(M_2)$ *iff* q_{01} *and* q_{02} *are indistinguishable in* M

DFA Minimization Algorithm

Consider a DFA $M = (Q, \Sigma, \delta, q_0, F)$

Goal: Minimize the number of states in M while preserving its language

Preparation:

Remove states that are unreachable from the start state

Partition the states into blocks of "equivalence states"

Algorithm for constructing minimized DFA N:

States of N are the blocks of the above partition

Start state of N is the block containing the start state of M

Accept states of N are the blocks containing accept states of M

Transition function γ is defined as follows:

For each block $S \subseteq Q$ and each input symbol $a \in \Sigma$, find the block T such that for each state $q \in S$, $\delta(q, a) \in T$ Why such a block T exists?

Define $\gamma(S, a) := T$

Proof of Correctness of DFA Minimization

From our construction, N is indeed equivalent to M.

But what if there is some another DFA K that is equivalent to M and has fewer states than N?

How to prove this does not happen? **Proof by contradiction!**

Assume for contradiction that there exists a DFA K that is equivalent to M and has fewer states than N.

Since K is equivalent to N (and M), it must also accept the same language

Thus, start states of N and K are indistinguishable

$p \equiv q \implies$ for each $a \in \Sigma$, $\delta(p, a) \equiv \delta(q, a)$

All states of N and K are reachable from the start state of N and K respectively

Each state of N is indistinguishable from at least 1 state of K

$\exists p_1, p_2 \in N$ that are indistinguishable for a state $p_3 \in K$ Pigeonhole Principle!

But then $p_1 \equiv p_3 \equiv p_2 \implies p_1 \equiv p_2$, a contradiction!