2-way Deterministic Finite Automata

Balaji,Sannidhi,Sasmita,Ramanan

IISc Bangalore

Outline

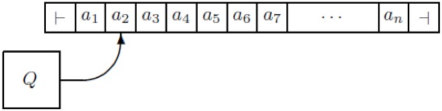
What is 2DFA?

2DFA

A Two-Way Deterministic Finite Automaton (2DFA) is a type of finite state machine that extends the capabilities of a regular Deterministic Finite Automaton (DFA) by allowing its read head to move bidirectionally along the input tape.

- 2DFAs were introduced in 1959 by Rabin and Scott.

How 2DFA works?



- Two-way Finite Automata has a read head, which can move left or right over the input string .
- Read head can revisit the input symbols any number of times.
- The input string is enclosed between left and right endmarkers $\vdash$ and $\dashv$, which are not elements of the input alphabet Σ.
- The read head will not move outside of the endmarkers.
- A 2DFA needs only a single accept state and a single reject state.

Turing Machine vs 2DFA

Turing Machine:

- Contains a read/write head that moves left or right along the tape
- Has unbounded memory.

2DFA:

- Has read only head that moves left or right along the tape.
- Has finite memeory like DFA.

Formal representation of 2DFA

A *2DFA* is of the form

$$(Q, \Sigma, \vdash, \dashv, \delta, s, t, r)$$

where,

- Q is a finite set of states.
- Σ is a finite set of input symbols.
- $\vdash$ is the left endmarker. ($\vdash \notin \Sigma$)
- $\dashv$ is the right endmarker. ($\dashv \notin \Sigma$)
- $\delta : Q \times (\Sigma \cup \{\vdash, \dashv\}) \to Q \times \{L, R\}$ is a transition function. (L = left, R = right)
- $s \in Q$ is the start state.
- $t \in Q$ is the accept state.
- $r \in Q$ is the reject state ($r \neq t$).

Properties of transition function

For all states p,

- $\delta(p, \vdash) = (q, R)$, for some $q \in Q$
- $\delta(p, \dashv) = (q, L)$, for some $q \in Q$

Current input symbol is $a \in \Sigma \cup \{\vdash\}$, $t =$ accept state, $r =$ reject state.

- $\delta(t, a) = (t, R)$ and $\delta(t, \dashv) = (t, L)$
- $\delta(r, a) = (r, R)$ and $\delta(r, \dashv) = (r, L)$

In general, $\delta(p, a) = (q, d)$ where $p, q \in Q$ and $d \in \{L, R\}$

Example 2DFA

2DFA for a^*

$(Q, \Sigma, \vdash, \dashv, \delta, s, t, r)$ where,

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{a, b\}$
- s = start state = q_0
- t = accept state = q_1
- r = reject state = q_2

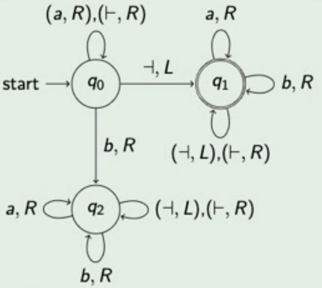


Table: Transition function δ

$\delta(,)$	$\vdash$	a	b	$\dashv$
q_0	(q_0, R)	(q_0, R)	(q_2, R)	(q_1, L)
q_1	(q_1, R)	(q_1, R)	(q_1, R)	(q_1, L)
q_2	(q_2, R)	(q_2, R)	(q_2, R)	(q_2, L)

Configurations

Fix an input $x \in \Sigma^*$. $x = a_1 a_2 a_3 \ldots a_n$. Let $a_0 = \vdash$ and $a_{n+1} = \dashv$.
$a_0 a_1 a_2 a_3 \ldots a_n a_{n+1} = \vdash x \dashv$.

Configuration

A configuration of the machine on input x is a pair (q, i) such that $q \in Q$ and $0 \leq i \leq n+1$. Informally, the pair (q, i) gives a current state and current position of the read head.
The start configuration is $(s, 0)$, meaning that the machine is in its start state s and scanning the left endmarker.

A binary relation $\xrightarrow[x]{1}$, the next configuration relation, is defined on configurations as follows:

$$\delta(p, a_i) = (q, L) \Rightarrow (p, i) \xrightarrow[x]{1} (q, i-1),$$

$$\delta(p, a_i) = (q, R) \Rightarrow (p, i) \xrightarrow[x]{1} (q, i+1).$$

Configurations

The relation $\xrightarrow[x]{1}$ describes one step of the machine on input x. We define the relations $\xrightarrow[x]{n}$ inductively, $n \geq 0$:

- $(p, i) \xrightarrow[x]{0} (p, i)$
- if $(p, i) \xrightarrow[x]{n} (q, j)$ and $(q, j) \xrightarrow[x]{1} (u, k)$, then $(p, i) \xrightarrow[x]{n+1} (u, k)$.

For any configuration (p, i), there is exactly one configuration (q, j) such that $(p, i) \xrightarrow[x]{n} (q, j)$.

Acceptance and Rejection

$$(p,i) \xrightarrow[x]{*} (q,j) \text{ iff } \exists n \geq 0 \text{ such that } (p,i) \xrightarrow[x]{n} (q,j).$$

Acceptance

The input x is accepted by the machine iff $(s,0) \xrightarrow[x]{*} (t,k)$ for some k.

Rejection

- The input x is rejected by the machine if $(s,0) \xrightarrow[x]{*} (r,k)$ for some k.

If the machine neither reaches accept state nor reject state then the machine is said to be looping on that input.
Language accepted by the machine $= \{x \in \Sigma^* | x$ is accepted by the machine$\}$.

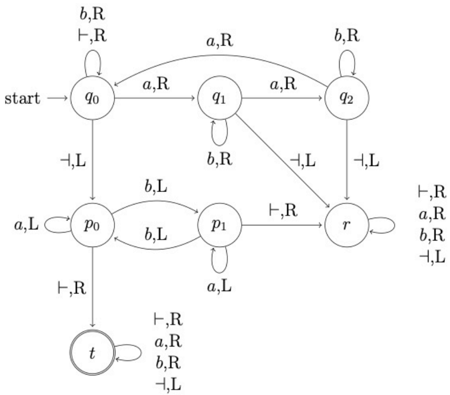
Constructing 2DFA 'M'

$L(M) = \{x \in \Sigma^* \mid \#a(x)$ is multiple of 3, $\#b(x)$ is multiple of 2$\}$

Machine Description

- Machine starts scanning from the left endmarker.
- Scan input string from left to right, counting only 'a's. If the count of 'a's is not a multiple of 3, reject and enter state r.
- Let q_0, q_1, q_2 be the states for counting 'a's.
  q_0: 3k; q_1: 3k+1; q_2: 3k+2.
- If the count of 'a's is a multiple of 3, start scanning from the right, counting only 'b's. If the count of 'b's is not a multiple of 2, enter state t; otherwise, enter state r.
- Let p_0, p_1 be the states for counting 'b's.
  p_0: 2k; p_1: 2k+1.

Constructing 2DFA 'M'



- input = bbaabab :

$q_0 \xrightarrow{\vdash} q_0 \xrightarrow{b} q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2 \xrightarrow{b} q_2 \xrightarrow{a} q_0 \xrightarrow{b} q_0 \xrightarrow{\dashv} p_0 \xrightarrow{b} p_1 \xrightarrow{a} p_1 \xrightarrow{b} p_0 \ldots.. \xrightarrow{\dashv} t$

DFA to 2DFA conversion

Theorem

- 2DFA only accepts regular languages.
- L(2DFA) = L(DFA) = Regular languages

Proof: L(DFA) $\subseteq$ L(2DFA)

For an arbitary DFA 'X', let us construct a 2DFA 'Y' that accepts the same language as X.

- X : (Q, Σ, δ, s, F)
- Let Y : (Q $\cup$ $\{$t, r$\}$, Σ, $\vdash$, $\dashv$, δ', s, t, r)
- δ': $\delta_R \cup$
  $\delta'((s, \vdash)) = (s, R)$
  $\delta'((f, \dashv)) = (t, L)$ for f $\in$ F ; $\delta'((n, \dashv)) = (r, L)$ for n $\in$ Q-F.
  $\delta'((t, a)) = (t, R)$; $\delta'((r, a)) = (r, R)$ for a $\in \Sigma - \{\dashv\}$
  $\delta'((t,\dashv)) = (t,L)$; $\delta'((r,\dashv)) = (r,L)$

DFA to 2DFA conversion

Proof: L(X) $\subseteq$ L(Y)

- Let x $\in$ L(X), len(x)=n.
- Then, $\widehat{\delta}$(s,x) = f where f $\in$ F
- $(s,0) \xrightarrow[\vdash x \dashv]{1} (s,1) \xrightarrow[\vdash x \dashv]{n} (f,n+1) \xrightarrow[\vdash x \dashv]{1} (t,n)$.
- As $(s,0) \xrightarrow[\vdash x \dashv]{*} (t,n+1)$, x $\in$ L(Y).
- Hence, L(X) $\subseteq$ L(Y).

Recall:
A configuration of the machine on input x is the pair (q, i) , which gives the current state and current position of the read head.
$(s, 0)$ means that the machine is in its start state s and scanning the left endmarker.

DFA to 2DFA conversion

Proof: L(Y) ⊆ L(X)

- Let x $\in$ L(Y), len(x)=n.
- Then (s,0) $\xrightarrow[\vdash x \dashv]{*}$ (t,m) in Y
- So (s,0) $\xrightarrow[\vdash x \dashv]{1}$ (s,1) $\xrightarrow[\vdash x \dashv]{n}$ (f,n+1) $\xrightarrow[\vdash x \dashv]{1}$ (t,n) .
- As $\widehat{\delta}$(s,x) = f where f $\in$ F, x $\in$ L(X)
- Hence, L(Y) $\subseteq$ L(X).

Therefore, L(Y) = L(X)

Language accepted by a 2-way DFA is regular

- Claim: The language accepted by a 2-way finite automaton is regular
- We will prove this using Myhill-Nerode theorem
- Recall that the Myhill-Nerode theorem states that a language is regular if and only if the canonical equivalence relation has finitely many equivalence classes

Language accepted by a 2-way DFA is regular

- Let $M = (Q, \Sigma, \vdash, \dashv, \delta, s, t, r)$ be a 2-way finite automaton
- Let $w = xz$ be a string in Σ^*
- As the automaton is 2-way, its read head can cross the boundary between x and z several times.
- Consider the function $T_x : (Q \cup \{\bullet\}) \to (Q \cup \{\perp\})$
- If the automaton goes to state q when it first crosses the boundary between x and z, define $T_x(\bullet) = q$
- If the read head never crosses the boundary between x and z, define $T_x(\bullet) = \perp$

Language accepted by a 2-way DFA is regular

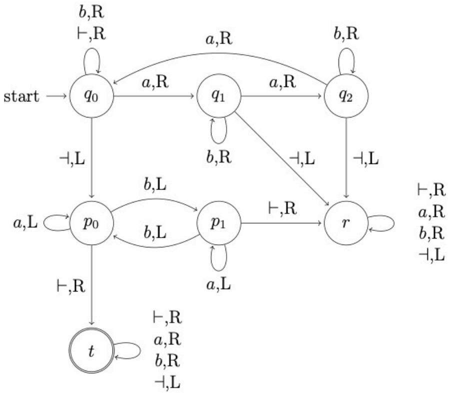
- Suppose the read head comes back from z to x and reaches state q
- Then it may either go back to z reaching state p, in which case define $T_x(q) = p$
- Else, it may never go back to z, in which case define $T_x(q) = \perp$
- Note that the function T_x is well-defined as the automaton is deterministic
- Also, T_x depends only on x and is independent of z
- And if y is another string in Σ^* such that $T_x = T_y$, then yz is accepted by the automaton if and only if xz is accepted by the automaton

Language accepted by a 2-way DFA is regular

- Hence, xRy if and only if $T_x = T_y$ becomes a canonical equivalence relation
- Since the number of unique functions is finite (at most $(k+1)^{k+1}$), the canonical equivalence relation has finitely many equivalence classes
- Hence the language accepted by the 2-way DFA is regular

Example

- Let us consider an example of a 2-way DFA accepting the language
  $L = \{x \in \{a, b\}^* | \#a(x) \text{ is a multiple of 3 and } \#b(x) \text{ is even}\}$



Example

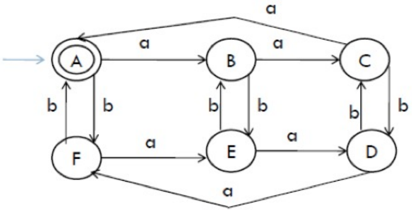
- For any $x \in \{a, b\}^*$,
- If $\#a(x) = k \mod 3$, then $T_x(\bullet) = p_k$
- If $\#b(x) = 0 \mod 2$, then $T_x(p_0) = t$ and $T_x(p_1) = r$
- If $\#b(x) = 1 \mod 2$, then $T_x(p_0) = r$ and $T_x(p_1) = t$
- And $T_x(t) = t$, $T_x(r) = r$
- Hence, the canonical equivalence relation has 6 equivalence classes and the language is regular

Constructing DFA from 2-way DFA

- Once we have the canonical equivalence relation, we can easily construct a DFA.
- Let $M = (Q, \Sigma, \vdash, \dashv, \delta, s, t, r)$ be a 2-way finite automaton
- Let Q' be the set of all equivalence classes of the canonical equivalence relation
- $s' = T_\epsilon$
- $\delta'(T_x, a) = T_x a$
- $F' = \{T_x | x \in L(M)\}$
- DFA $M' = (Q', \delta', s', F')$ will accept the same language as the 2-way DFA M

Example

- The DFA accepting the same language as the 2-way DFA is as follows:



Unary finite automata

Below, H(n) represents the function $e^{\sqrt{nlog(n)}}$

These are some statements related to unary finite automata

- Each unary n-state 2dfa can be simulated by a 1dfa with O(H(n)) states.
- For each n there is a unary n-state 2dfa A such that each 1dfa recognizing L(A) requires Ω(H(n)) states.

Unary finite automata

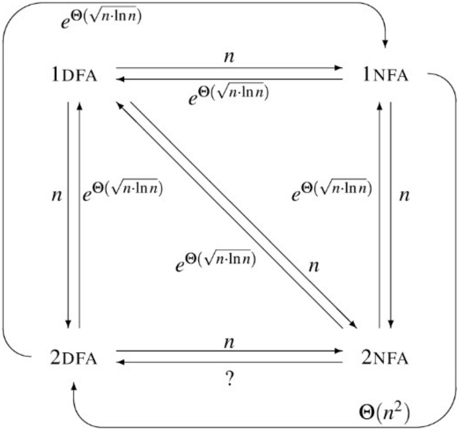
- For each n there is a unary n-state 2dfa A such that each 1nfa recognizing L(A) requires Ω(H(n))
- Each unary n-state 1nfa A can be simulated by a 2dfa B with $O(n^2)$ states.
- For each n there is a unary n-state 1nfa A such that each 2dfa recognizing L(A) requires $\Omega(n^2)$ states.

Unary finite automata

Informally speaking we can see that 2dfa's are hard to simulate by ldfa's, even if we consider only unary languages.

Also, for unary languages,two-way motion is more powerful, in a sense, because we can simulate unary lnfa's by 2dfa's increasing the number of states only polynomially, which is not possible the other way round.

Unary finite automata



1nfa to 2dfa

Lemma A: If gcd(a,b)=1, then the greatest number such that the equation $ax + by = c$ has no solutions in natural numbers is (a-1)(b-1)-1.

Theorem:For each n there is a unary n-state 1nfa A such that each 2dfa recognizing L(A) requires $\Omega(n^2)$ states.

Proof: Let L=$\{x|x = nx_1$+(n-1)x_2 for x_1, x_2 $\in$N$\}$. L can be recognized by a Infa A with n states. A=(Q, q_0, E, F) is defined as follows: Q=q_0,..., q_{n-l}, E $= \{(q_i,\ q_{i+1})|i = 0, ..., n-1\} \cup (q_1, q_3)$ (the addition is mod n) and F = q_0. Let m = max($\mathbb{N}$-L). By Lemma A, m = O(n^2). Consider a 2dfa B recognizing L and its computation on m. Suppose that, in all passes on m, B enters a cycle and let y_1,...,y_k be the lengths of these cycles. Then B would reject also m'= m + lcm($y_1, .., y_k$), which contradicts the fact that m' $\in$ L. Therefore, there is a pass. of B on m without a cycle and the theorem follows.

(Unary)2-DFA to 1-NFA/1-DFA - A worst case example

- Let $F(n)$ be defined as follows:
- Maximum value of $lcm(x_1, x_2, \ldots x_k)$ such that $x_1 + x_2 + \ldots + x_k = n$ and $x_1, x_2, \ldots x_k \in \mathbb{N}$

Some properties of $F(n)$:

- $x_1, x_2, \ldots x_k$ can be found to be coprime satisfying $lcm(x_1, x_2 \ldots x_k) = F(n)$
- $F(n) = O(H(n))$

(Unary)2-DFA to 1-NFA/1-DFA - A worst case example

- Let the language $L = \{a^{iF(n)} \mid i \in \mathbb{N}\ , i \geq 1\}$.
- Take any 1-NFA accepting this language.
- Any simple path from a start state to a final state must have length at least $F(n)$.
- Canonical MN Relation has atleast $F(n)$ classes.
- But 2-DFA can do better. Have x_i states to check divisibility by x_i.
- Upon seeing right end marker transition to new set of x_{i+1} states if divisible by x_i.
- Reject if not divisible by any x_i. Go to final state if divisible by all x_i. Total number of states $= n + 2$.

(Unary)2-DFA to 1-DFA - Best known Bound

- Any unary 2-DFA with can be converted to 1-DFA with at most $O(H(n))$ states.
- Any 2-DFA can be converted into an equivalent sweeping 2-DFA without changing the number of states. [Chrobak]
- For accepting words of length $\leq n$ accepted by the 2-DFA, we have n states, mark them as final accordingly
- Note that any word of length longer than n accepted by the 2-DFA must pass through a cycle/pump.
- Any given state can be a part of at most one of these cycles.
- Say on a word u of length more than n the machine passes through at most k cycles.

(Unary)2-DFA to 1-DFA - Best known Bound

- Let the lengths of the cycles be $y_1, y_2, \ldots y_k$. Then $y_1 + y_2 + \ldots y_k \leq n$.
- If a machine accepts a word u of length more than n, it must also accept a word of length $|u| + lcm(y_1, y_2, \ldots y_k)$.
- Construct a 1-DFA with first n states as mentioned and a loop of length $lcm(y_1, y_2, \ldots y_k)$ attached to the last state. Mark states as final accordingly.
- Prove: This accepts the same language as the 2-DFA.
- The number of states in the 1-DFA is at most $n + lcm(y_1, y_2, \ldots y_k) \leq n + F(n) = O(H(n))$.

Some other automata

- **2 way Non-deterministic Finite Automata:-** A two-way nondeterministic finite automaton (2NFA) may have multiple transitions defined in the same configuration.
- Its transition function is $\delta : (Q \times \Sigma \cup \{L, R\}) \rightarrow 2^{Q \times \{left, right\}}$
- A 2NFA accepts a string if at least one of the possible computations is accepting. Like the 2DFAs, the 2NFAs also accept only regular languages.
- **Sweeping Automata:-** A two-way automaton performing head reversal only when the input head is visiting the endmarkers is called sweeping automaton.
- **Rotating Automata:-**A computation of a rotating automaton is a sequence of left-to-right scans of the input. In particular, when the right end of the input is reached, the computation continues on the leftmost input symbol.

Some other automata

- In other words, we can imagine the input tape as circular, with a special cell containing a marker and connecting the end with the beginning of the tape.
- With a trivial transformation which doubles the number of the states, each rotating automaton can be transformed into an equivalent sweeping automaton.
- **2 way Alternating Finite Automata(AFA):** Structure same as 2 way NFA. Transitions divided into 2: existential and universal.
- Existential and universal transitions simulate all possible moves that can be made. Every state is given a truth value.
- Even if there is one simulation that leads to a final state, existential transitions return 1. Universal transitions return 1 only if all simulations lead to a final state.

Other established results and open problems

- Christos Kapoutsis determined that transforming an n-state 2-DFA to an equivalent 1-DFA requires $n(n^n - (n-1)^n)$ states in the worst case.
- If an n-state 2DFA or a 2NFA is transformed to a 1-NFA, the worst-case number of states required is $\binom{2n}{n+1} = O(\frac{4^n}{\sqrt{n}})$.
- Sipser constructed a sequence of languages, each accepted by an n-state NFA, yet which is not accepted by any sweeping automata with fewer than 2^n states.
- **Sakoda-Sipser's open problem:**- Is there a 2-DFA with polynomial number of states accepting the same language as a 2-NFA?
- **Another open problem:** What is the relationship between unary 1-afa's (or 2-afa's) and other fa's? It is easy to show some lower and upper bounds for 1-afa's with only universal states.

References

- Papers referenced are available on Github
- Wikipedia link