



Deterministic finite automata and regular languages

1. Deterministic finite automata
2. Regular languages and regular expressions
3. Nondeterministic finite automata
4. Proving a language is not regular
5. A foundation for lexical analysis

1. Deterministic finite automata

- A *transition system* is a labeled digraph, where vertices denote *state* (memory), edges denote transitions and labels
- 1-state example:  2-state example: 
- DFA differs from *random-access machine* (RAM) model, in which state is a value assignment to a set of variables

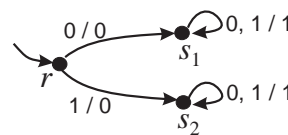
Transition systems

- Components:
 - Alphabet Σ (set of symbols)
 - State set (a state can be anything)
 - Transition function or relation
(rules for going from one state to another)
- Corresponds to a graph, labeled with symbols that trigger transitions
- DFA has finite set of states, unique destination for any transition

Graph of a transition function

- A *graph* $G = (V, E)$ is a set of vertices and a set of ordered pairs of vertices
- E is a *relation* on V
- If edges are *labeled* with symbols, and each edge from vertex u labeled with symbol a goes to a unique vertex v ...
- ... then the labeled graph denotes a *transition function*

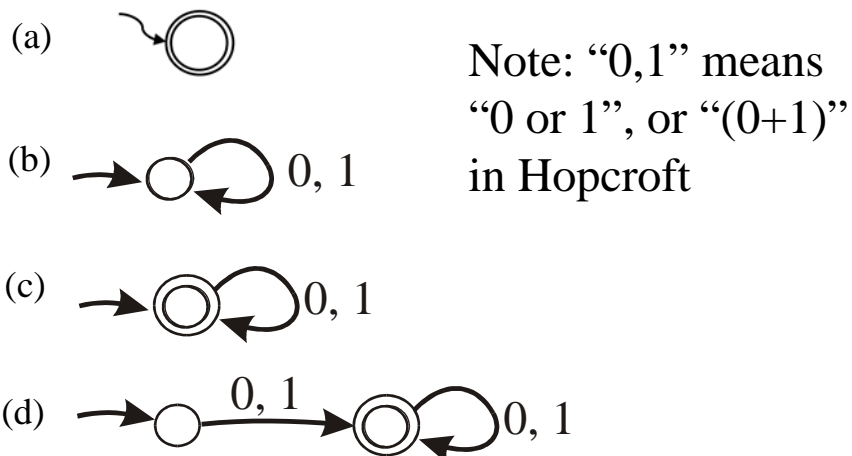
$$\delta : V \times E \rightarrow V$$



Definition: DFA

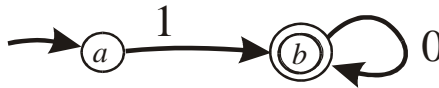
- A DFA is a 5-tuple $\langle Q, \Sigma, \delta, q_0, F \rangle$ where
 - Q is a set of states
 - Σ is a finite alphabet
 - $\delta: Q \times \Sigma \rightarrow Q$ is a state-transition function
 - $q_0 \in Q$ is the starting state
 - F is the set of accepting states
- We can also define $\delta^* : Q \times \Sigma^* \rightarrow Q$, the reflexive transitive closure of δ , which tells what state δ yields for a *string*

Some simple DFAs



Language of a DFA

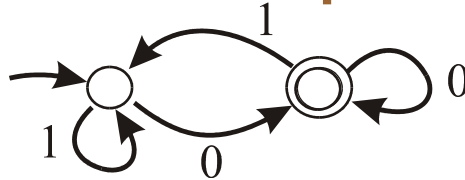
- For DFA M , the language of M , $L(M) = \{ x \in \Sigma^* \mid (\delta^*(q_0, x) = q') \wedge (q' \in F) \}$
- ...where Σ^* is the set of all strings over Σ , and for string x , $\delta^*(q_0, x)$ is the state reached after repeated applications of δ to states and to elements of x
- *Example:* $L(M)$ for M below is all bit strings that have a '1' followed by 0 or more '0's.



Regular languages

- *Language:* a set of sequences over a finite alphabet of symbols
- *Regular language:* a language whose elements are all recognized by some DFA
- The following decision problems are equivalent:
 - Is sequence x in regular language L ?
 - Is x accepted by a DFA A , where $L(A) = L$?
- Let \mathcal{RL} be the set of regular languages

Example DFA

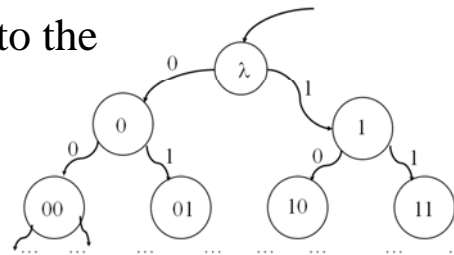


- This DFA accepts strings that start with any number of 1's (possibly none), followed by a 0, followed by any number of (01)s, followed by any number of 0s
- Some elements of its language:
0, 10, 00, 100, 110, 010, 1010, 10100

Thm: Finite languages are regular

Proof by construction:

- For any finite language L , construct a DFA whose transition paths form the tree corresponding to all the strings in L
- Each vertex has up to $|\Sigma|$ out-transitions
- Edges corresponding to the last symbol of a word in L go to final states



2. Regular languages and regular expressions

- Any RL may be specified by a *regular expression* using any combination of these operations:
 - Concatenation
 - Selection ($|$, $+$, \cup)
 - Iteration ($*$)
- *Examples* (what are their DFAs?):
 - $(01)^*$: all strings that repeat the string 01, zero or more times
 - $0^* | 10^*$: all strings that either consist of a 0 followed by zero or more 1's, or consist of a 1 followed by zero or more 0's

Complement of a language

- *Complement of L* is all strings not in L
Examples: Complement of Σ^* is \emptyset ,
 complement of 0^* is $(0+1)^* 1 (0+1)^*$
- *Theorem:* The complement of a regular language is regular.
- *Proof:* Construct a DFA that accepts all inputs *not* in L , starting with a DFA that recognizes L ; make all accepting states be non-accepting, all non-accepting states accepting

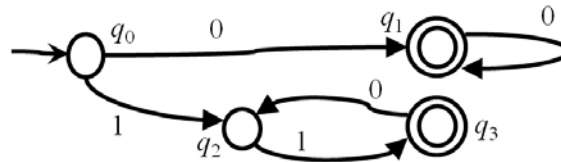
DFA \rightarrow RE

Cases:

- *Sequence* of states with transitions:
Concatenate REs (E_1E_2)
- *Branch* from one state to two or more others:
 $E_1 + E_2 + \dots + E_n$
- *Loop* (transition from one state to the same or a preceding state): Star the RE for the path from destination of transition to its origin (E^*)

A method for DFA \rightarrow RE

- *Observation:* Any state q of a DFA may be named for a regular expression that expresses the paths that take M from start state to q



$$q_0 = \lambda$$

$$q_1 = 00^*$$

$$q_2 = 1(01)^*$$

$$q_3 = 11(01)^*$$

Reg. expr. for this DFA is $(q_1 \cup q_3)$

3. Nondeterministic finite automata

- *Definition:* NFA $A = \langle Q, \Sigma, \delta, q_0, F \rangle$, s.t. $\delta: Q \times \Sigma \rightarrow 2^Q$ permits transitions to any of several states from a given state on a given symbol, and
- δ may have λ -transitions (transitions without input)
- A transition from q to q' on a , with $q, q' \in Q, a \in \Sigma$, denotes that the NFA *can* go from q to q' on a

λ -transitions

- We may consider any state q to have a language, i.e., $L(q) = \{x \in \Sigma^* \mid \delta^*(q, x) \in F\}$
- Now, if q has a λ transition to r , then $L(r) \subset L(q)$, because any string can take M from r to the same states as from q
- Let $\lambda\text{-closure}(q) = \{q\} \cup \lambda\text{-closure}(\{r \mid r \in \delta(q, \lambda)\})$

Application of NFAs

- For languages L_1 and L_2 , the concatenation L_1L_2 is $\{xz \mid x \in L_1, z \in L_2\}$
- *Theorem:* $(L_1, L_2 \in RL) \Rightarrow (L_1L_2 \in RL)$
- *Proof:* DFAs M_1 and M_2 may be concatenated using λ -transitions as follows:



- This NFA recognizes the languages of M_1 and M_2 , concatenated, $L(M_1)L(M_2)$

Theorem: one accept state

- *Thm:* For any DFA, there is an NFA with only one accept state that recognizes the same language
- *Proof:* Construct the NFA from the DFA, keeping one accept state q_{acc} and such that each other acceptance state
 - becomes a reject state, and
 - has a λ transition to q_{acc}

Theorem: DFA \rightarrow NFA

- *Theorem:* Every regular language is recognized by some NFA
- *Proof:* A DFA is by definition an NFA without nondeterministic transitions

Reverses of RLs are regular

Theorem: If L is regular, then *reverse* of L , L^R (set of elements of L each spelled backwards), is regular

Proof: Construct a DFA that accepts L^R :

1. start with a one-accept-state DFA that recognizes L
2. reverse the directions of all transitions
3. swap accepting and starting states
4. new start state is a new state with λ transitions to each old accept state

RE \rightarrow NFA

- *Theorem:* For any regular expression, an NFA may be constructed that recognizes the same language
- *Proof (by construction):*
 - for concatenated parts of RE, a sequence of states
 - for “+”, a fork
 - for “*”, a transition to first state in starred part of RE
- *Example:* $(0 \mid 1)^* 1$ (strings that end in 1)

NFA \rightarrow DFA (subset construction)

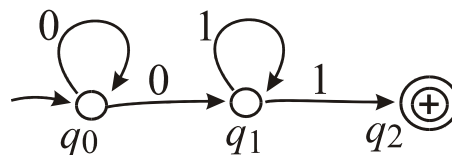
- To construct a DFA $M' = \langle Q', \Sigma, \delta', q_0', F' \rangle$ from an NFA $M = \langle Q, \Sigma, \delta, q_0, F \rangle$, let $Q' = 2^Q$ (set of all subsets of Q).
- Then derive δ' from δ by merging all states with λ -transitions between them...
- and let $\delta'(q, a) = \bigcup_{r \in q} \{ \delta(r, a) \}$ (the state of M' that is the set of all states of M to which there is a transition on a from a state in q)
- F' is the set of states of M that contain *some* state in F
- $q_0' = \{q_0\}$

Intuition for the subset construction

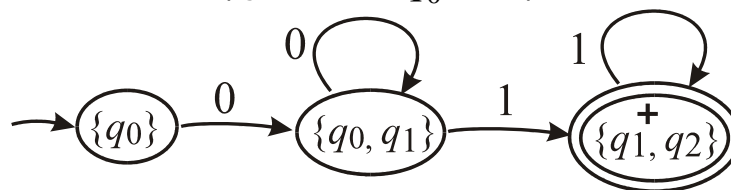
- The DFA has states that are sets of NFA states, because the NFA transition function is from states to sets of states
- λ -transitions mean that states involved in them are in effect equivalent, so they merge in the DFA M'
- $\delta'(q, a)$ of DFA M' is the set of states the NFA M can go to from some state that is in q of DFA M' .

NFA \rightarrow DFA example

NFA $M = \langle Q, \Sigma, \delta, q_0, F \rangle$, $L(M) = 0^+1^+$

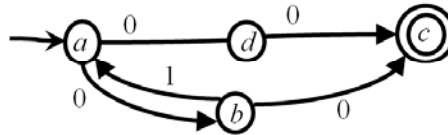


DFA $M' = \langle Q', \Sigma, \delta', q_0', F' \rangle$, $L(M') = 0^+1^+$

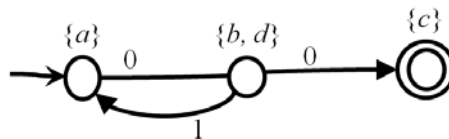


Subset construction example

- NFA $M = \langle Q, \Sigma, \delta, q_0, F \rangle$



- $L(M) = (01)^*00$
- Construct DFA $M' = \langle Q', \Sigma, \delta', q_0', F' \rangle$



Significance of NFA \rightarrow DFA

- Every NFA can be converted to a DFA, and every DFA is an NFA
- This means that NFAs have the same computational power as DFAs (recognize the same languages)
- So the following are equivalent:
 - Some NFA recognizes language L
 - Some DFA recognizes L
 - L is regular
 - L is generated by a regular expression

4. The Pumping Lemma

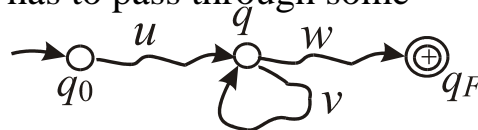
Lemma: A property of infinite regular languages L is:

$$(x \in L) \Rightarrow (x = uvw) \text{ s.t. } (\forall k \in \mathbf{N}) uv^k w \in L, |v| > 0$$

(Any string in the language can be expressed as the concatenation of three strings, the middle of which can be “pumped” any number of times without leaving the language)

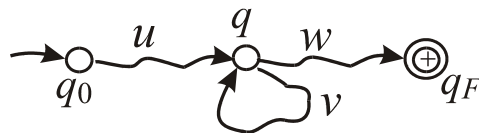
Proof:

1. Consider RL L and DFA M , s.t. $L = L(M)$,
 $M = \langle Q, \Sigma, \delta, q_0, F \rangle$, and L is infinite
2. Then for some x in L , $|x| > |Q|$
3. So when M inputs x , M has to pass through some state q more than once



Pumping Lemma

4. Hence there exists $n_0 \leq |Q|$ s.t. if $\delta^*(q_0, u) = q$,
 $\delta^*(q, w) = q_F$, $uvw \in L$, $|v| > 0$, and $|uvw| \geq n_0$,
then $uv^k w \in L$ for any $k \in \mathbf{N}$.



Pigeonhole Principle is applied here

Proving a language is not regular

Theorem: If $L = 0^n 1^n$ then $L \notin \mathcal{RL}$

Proof: By **Pumping Lemma**, if L is regular then $(\forall x \in L)$
 $(\exists u, v, w)$ s.t. $(uvw = x \wedge |v| > 0 \wedge (\forall k) uv^k w \in L)$

Cases:

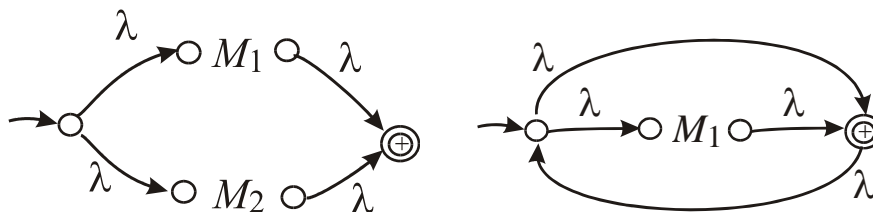
- If v were all 0's then we could "pump" it with a large enough k so that there are more 0's than 1's so v is not all 0's; similarly v can't be all 1's
- If v is 0's and 1's, then pumping even once produces a string not in L
- Hence no v can exist that satisfies the Pumping Lemma
- Therefore L is not regular

Closure properties of RLs

Theorem: For two regular languages, L_1 and L_2

- $L_1 L_2$ is regular (concatenation)
- $(L_1 | L_2)$ is regular (union)
- $(L_1)^*$ is regular (Kleene star)

Proof of (b) and (c):

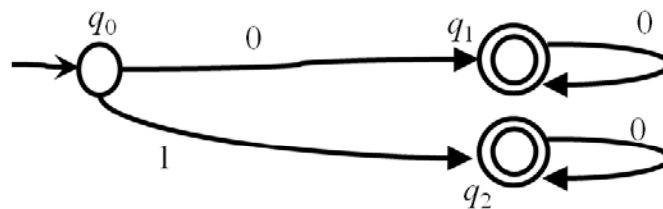


Minimal DFA for a RL

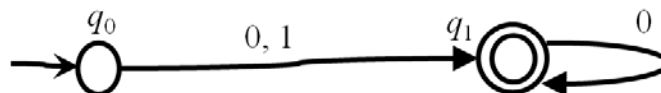
- *Theorem:* For any RL L , there exists a *unique** DFA M s.t. $L(M) = L$ and M has no more states than any other DFA M' where $L(M') = L$ (Myhill-Nerode)
- There exists an algorithm that reduces a DFA to its minimal version
- The algorithm proceeds by finding indistinguishable states and merging them

—
* unique up to isomorphism (renaming of states)

Minimization example



- In the DFA above, states q_1 and q_2 are indistinguishable
- The DFA below is minimal for the same language



5. A foundation for lexical analysis

- *Lexical analysis* (tokenization) is a precondition for parsing
- To recognize identifiers, numerals, operators, etc., implement a DFA in code
- *State* is an integer variable, δ is a *switch* statement
- Upon recognizing a lexeme, return its lexical class and restart DFA with next character in source code

References

Daniel I. I. Cohen. *Introduction to Computer Theory*, 2nd ed. Wiley, 1997.

J. Hopcroft, R. Motwani, J. Ullman. *Introduction to Automata Theory, Languages, and Computation*, 3rd Ed. Addison-Wesley, 2007.

See also *JFLAP* manual.