

Lecture 2: Finite Automata

David L. Dill

Department of Computer Science

Outline

- Introduction to regular languages.
- Deterministic finite automata (DFA)
- Nondeterministic finite automata (NFA)
- Proof that NFA and DFA languages are the same (subset construction).

Motivation

The next few lectures will be about *Regular languages*.

Regular languages can be infinite, and can be described in several ways: by deterministic or non-deterministic finite automata, or by regular expressions.

Regular languages are of great practical as well theoretical importance.

- Widely used in applications (compilers, pattern matching, specification and formal verification of systems).
- Constructions from automata theory are used in these applications.
- Basis for more sophisticated representations (automata on infinite strings, tree automata, timed automata).
- Decidable – Many problems on finite automata are solvable. (membership, emptiness, universality, subset, etc.)

We'll have lots of *positive* results with regular languages.

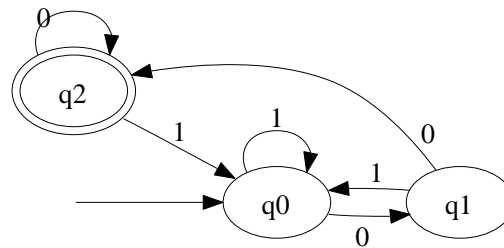
Later, the most important results will be negative.

“More expressiveness” leads to “harder problems.”

Deterministic Finite Automata

A finite automaton is a mathematical model of a machine that reads strings and says “yes” or “no” for each string.

A DFA defines an infinite language: the set of strings for which it says “yes.”



Deterministic Finite Automata

A DFA is a quintuple: $(Q, \Sigma, q_0, \delta, F)$ where

- Q is a finite set of states.
- Σ is an alphabet.
- $q_0 \in Q$ is a start state.
- $\delta: Q \times \Sigma \rightarrow Q$ a next-state function.
- A set of final states $F \subseteq Q$.

(Students: make sure you know what all these are in the diagram on the previous slide.)

Language of a DFA

We extend the next state function δ to work on whole strings. For our example DFA in slide 4,

$$\begin{aligned}\hat{\delta}(q_1, \epsilon) &= q_1 \\ \hat{\delta}(q_0, 010101000) &= q_2 \\ &\dots\end{aligned}$$

Recursive Definition of $\hat{\delta}$

Base: $\hat{\delta}(q, \epsilon) = q$

Induction: $\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a)$ where $x \in \Sigma^*$ and $a \in \Sigma$.

Def A DFA *accepts* a string $x \in \Sigma^*$ iff $\hat{\delta}(q_0, x) \in F$.

Def The language of a DFA D (written $L(D)$) is the set of strings accepted by D . $L(D) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F\}$

Question: What is the language of the example DFA?

Non-determinism

In a DFA, from each state

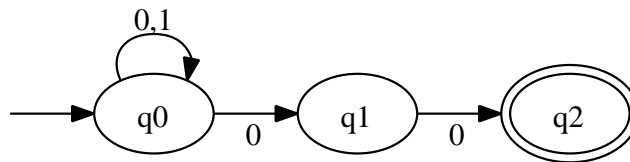
1. each input symbol leads to exactly one next state, and, therefore,
2. each input string leads to a unique state.

In an NFA,

1. there can be several possible next states for each input symbol, and
2. an input string can lead to several states (or *no* states).

The string is accepted by an NFA if *at least one of these states is accepting.*

Here is an example:



Essence of non-determinism

- Multiple alternative computations (often, huge numbers of them).
- Success if *any* succeed.

Different perspectives on non-determinism

- Massive parallel search.
- Search with “backtracking”
- “Guessing” the right option at each step.

Non-deterministic Finite Automata

An NFA is a 5-tuple $(Q, \Sigma, q_0, \delta, F)$.

Everything is the same as a DFA, *except* δ .

DFA: $\delta: Q \times \Sigma \rightarrow Q$.

$\delta(q_1, 0) = q_1$ – it returns a unique state

NFA: $\delta: Q \times \Sigma \rightarrow 2^Q$.

2^Q is “powerset of Q ”, which is the set of all subsets of Q .

$\delta(q_0, 0) = \{q_0, q_1\}$ – it returns a set of states,

$\delta(q_1, 1) = \emptyset$ – this set can be empty too.

NFA Acceptance

Def: $\hat{\delta}$ for NFAs

base: $\hat{\delta}(q, \epsilon) = \{q\}$

induction: $\hat{\delta}(q, xa) = \bigcup_{s \in \hat{\delta}(q, x)} (\delta(s, a))$

Language of an NFA

$L(N) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$.

Example: From the NFA in slide 7, verify that

$$\hat{\delta}(q_0, 10100) = \{q_0, q_1, q_2\}$$

$$\hat{\delta}(q_1, 01000) = \emptyset$$

...

(From the start state, does the string lead to at least one final state?)

NFAs vs. DFAs

Amazingly, an NFA is no more powerful than a DFA (although it can sometimes be a lot smaller).

(With other kinds of automata, the non-deterministic versions are sometimes more powerful than the deterministic ones.)

Theorem For every NFA N , there is a DFA D such that $L(N) = L(D)$.

Below, N will represent an arbitrary NFA, and D will be the DFA resulting from applying the subset construction to N .

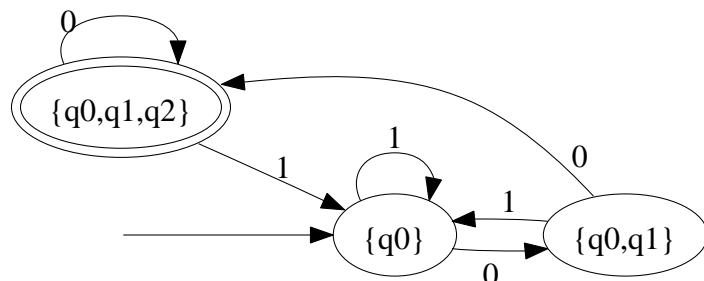
The proof relies on the *subset construction*.

Subset Construction Example

The D tracks the possible set of states that N can be in after reading a particular string.

Hence, a state in $D =$ a set of states in N .

Here is a tracking DFA for the NFA from slide 7



The reasoning behind the construction was as follows:

... Suppose the original automaton was in one of the states in the set $\{q_0, q_1\}$, and we received a 0, what are all the states we can reach?

Clearly the answer is $\{q_0, q_1, q_2\}$.

Therefore there is an arrow in the DFA from the *dfa state* $\{q_0, q_1\}$ to the state $\{q_0, q_1, q_2\}$.

Subset Construction

Given $N = (Q, \Sigma, q_0, \delta_N, F_N)$

construct $D = (2^Q, \Sigma, \{q_0\}, \delta_D, F_D)$ such that $L(D) = L(N)$.

The key is in the definition of δ_D and F_D :

where $\delta_D(S, a) = \bigcup_{q \in S} \delta_N(q, a)$, for each $S \in 2^Q$.

The final states of the D are the subsets that contain at least one final state of N

$$F_D = \{S \in 2^Q \mid S \cap F_N \neq \emptyset\}$$

Proof of the subset construction

Here is what we want:

Theorem For every NFA N , if D is the DFA obtained by the subset construction, $L(N) = L(D)$.

But it's actually easier to prove a stronger claim:

Lemma $\hat{\delta}_D(\{q_0\}, w) = \hat{\delta}_N(q_0, w)$ for all $w \in \Sigma^*$

(I.e., the *state* you get to by running D on w is the same as the *set of states* you get to by running N on w .)

Reminders: DFA $\hat{\delta}$:

Base: $\hat{\delta}(q, \epsilon) = q$

Induction: $\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a)$

NFA $\hat{\delta}$:

Base: $\hat{\delta}(q, \epsilon) = \{q\}$

Induction: $\hat{\delta}(q, xa) = \bigcup_{s \in \hat{\delta}(q, x)} (\delta(s, a))$

Subset construction proof, cont

Lemma $\hat{\delta}_D(\{q_0\}, w) = \hat{\delta}_N(q_0, w)$ for all $w \in \Sigma^*$

Proof We prove by induction on the structure of w .

Base: if $w = \epsilon$, we have

$$\begin{aligned}\hat{\delta}_D(\{q_0\}, \epsilon) &= \{q_0\} && \text{def of } \hat{\delta} \text{ for DFAs} \\ &= \hat{\delta}_N(q_0, \epsilon) && \text{def of } \hat{\delta} \text{ for NFAs}\end{aligned}$$

Induction: Let $w = xa$, where x is any string and a any symbol.

$$\begin{aligned}\hat{\delta}_D(\{q_0\}, xa) &= \delta_D(\hat{\delta}_D(\{q_0\}, x), a) && \text{def of } \hat{\delta} \text{ for DFAs} \\ &= \delta_D(\hat{\delta}_N(q_0, x), a) && \text{ind. hyp.} \\ &= \bigcup_{q \in \hat{\delta}_N(q_0, x)} \delta_N(q, a) && \text{def of } \delta_D \\ &= \hat{\delta}_N(q_0, xa) && \text{def of } \hat{\delta}_N\end{aligned}$$

Proof of the subset construction (continued)

It's now easy to prove the theorem we actually wanted.

Theorem $L(D) = L(N)$.

Proof Let x be any string in Σ^* .

$L(N) = L(D)$ is equivalent to saying that x is accepted by D iff x is accepted by N . That is what we prove.

$x \in L(D)$	\Leftrightarrow	$\hat{\delta}_D(\{q_0\}, x) \in F_D$	Def of accept in DFA
	\Leftrightarrow	$\hat{\delta}_D(\{q_0\}, x) \cap F_N \neq \emptyset$	Def of F_D in subset construction
	\Leftrightarrow	$\hat{\delta}_N(q_0, x) \cap F_N \neq \emptyset$	the previous Lemma
	\Leftrightarrow	$x \in L(N)$	Def of accept in NFA

Equivalence of NFAs and DFAs

Theorem: A language L is $L(N)$ for some NFA N iff if it is $L(D)$ for some DFA D .

Proof: The subset construction was the hard part of this: Given any NFA, a DFA accepting the same language can be constructed. So, whenever a language is accepted by an NFA, it is also accepted by some DFA.

The other direction of the proof is obvious. Every DFA is essentially an NFA. The only change is that $\delta_N(q, a) = \{\delta_D(q, a)\}$. All other aspects of the FA remain the same, including F .

Subset Construction Efficiency

In the worst case, the subset construction generates $2^{|Q|}$ states.

In practice, it is often unnecessary to generate them all, since few are actually *reachable* by any input string.

A more efficient procedure: Generate the subset *on the fly*. Create them as you trace through the NFA. That way, you only generate the *accessible* subsets.

Example: Notice that in our subset construction example in the previous lecture the DFA states for the sets $\{q_0, q_2\}$ and the \emptyset are absent in the final DFA obtained.

Clearly, we used the efficient procedure to compute it!