Lecture 2: Finite Automata

David L. Dill and Sriram Sankaranayanan

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.

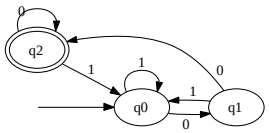
Regular languages are of great practical as well theoretical importance.

- Widely used in applications (compilers, pattern matching, specification and formal verification of systems).
- Constructions from automata theory are used in these applications.
- Robust – many sets turn out expressable as FA, and many representations are equivalent, they are closed under many operations.
- 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.)

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.”



DFA- Example

The tuple for the example DFA in slide 4 is

$$
\begin{aligned}
Q &= \{q_0, q_1, q_2\},\\
\Sigma &= \{0, 1\},\\
q_0 &= q_0\\
\delta &:
\end{aligned}
$$

Q	0	1
q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_2	q_0

$$
F = \{q_2\}
$$

Deterministic Finite Automata

A DFA consists of five parts:

- A finite set of states (Q).
- An alphabet Σ.
- A start state $q_0 \in Q$.
- A next-state function δ: $Q \times \Sigma \rightarrow Q$.
- A set of final states $F \subseteq Q$.

In mathematese, we would say "A DFA is a 5-tuple" $(Q, \Sigma, q_0, \delta, F)$ where Q is ..."

Non-determinism

In a DFA, from each state

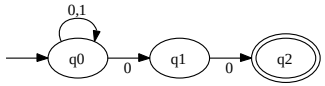
1. each input symbol leads to exactly one next state, and, therefore,
2. each input string leads to a unique state.

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:



Language of a DFA

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

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

Recursive Definition of $\hat{\delta}$

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

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

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

Question: What is the language of the example DFA?

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

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: $\delta\colon Q \times \Sigma \to Q$.

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

NFA: $\delta\colon Q \times \Sigma \to 2^Q$.

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.

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.

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

The proof relies on the *subset construction*.

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

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

Example: From the NFA in slide 8, verify that

$$\hat{\delta}(q_0, 10100) = \{q_0, q_1, q_2\}$$
$$\hat{\delta}(q_1, 01000) = \emptyset$$
$$\ldots$$

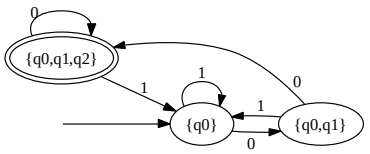
(From the start states, does the string lead to at least one final state.)

Subset Construction Example

The DFA produced by the subset construction *tracks* the possible set of states that the NFA can be in.

Hence, a state in the DFA = a set of NFA states.

Here is a tracking DFA for the NFA from slide 8



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 $\{q_0, q_1, q_2\}$.