Finite Automata and Formal Languages

TMV027/DIT321– LP4 2013

Lecture 5
Ana Bove

April 11th 2013

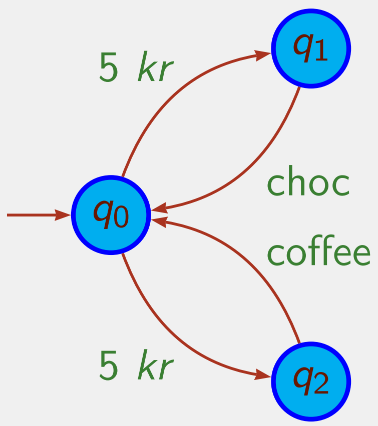
Overview of today's lecture:

- Non-deterministic Finite Automata;
- Equivalence between DFA and NFA.

Non-deterministic Finite Automata

A non-deterministic finite automata (NFA) can be in several states at once.

That is, given a state and the next symbol, the automata can "move" to many states.



Intuitively, the vending machine can *choose* between different states.

Observe that we do not need a *dead* state here.

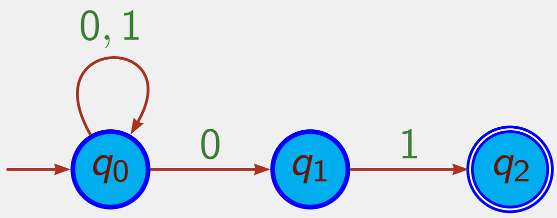
When Does a NFA Accepts a Word?

The automaton accepts w iff there is *at least one* computation path starting from the start state to an accepting state.

Intuitively, the automaton can *guess* a successful computation if there is one.

Example:

NFA accepting words that end in 01

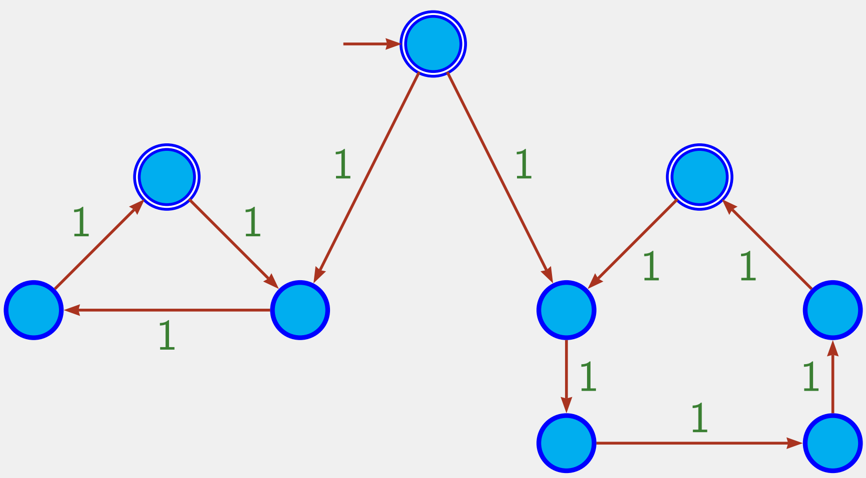


What are all possible computations for the string 10101?

Will 10101 be accepted by the NFA?

NFA Accepting Words of Length Divisible by 3 or by 5

Let $\Sigma = \{1\}$.



The automaton *guesses* the right direction and then verifies that $|w|$ is correct!

What would be the equivalent DFA?

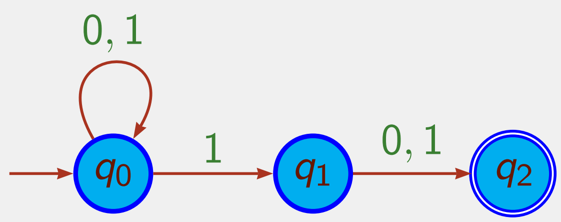
Non-deterministic Finite Automata

Definition: A *non-deterministic finite automaton* (NFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ consisting of:

1. A finite set Q of *states*;
2. A finite set Σ of *symbols* (alphabet);
3. A *transition function* $\delta : Q \times \Sigma \to \mathcal{P}ow(Q)$
   ("partial" function that takes as argument a state and a symbol and returns a *set of states*);
4. A *start state* $q_0 \in Q$;
5. A set $F \subseteq Q$ of *final* or *accepting* states.

Example: NFA

Let us define an automaton accepting only the words such that the second last symbol from the right is 1.



δ	0	1
$\to q_0$	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\{q_2\}$	$\{q_2\}$
$*q_2$	$\emptyset$	$\emptyset$

The automaton *guesses* when the word finishes.

Extending the Transition Function to Strings

As before, we want to be able to determine $\hat{\delta}(q, x)$.

We define this by recursion on x.

Definition:

$$\hat{\delta} : Q \times \Sigma^* \to \mathcal{P}ow(Q)$$
$$\hat{\delta}(q, \epsilon) = \{q\}$$
$$\hat{\delta}(q, ax) = \bigcup_{p \in \delta(q,a)} \hat{\delta}(p, x)$$

That is, if $\delta(q, a) = \{p_1, \ldots, p_n\}$ then

$$\hat{\delta}(q, ax) = \hat{\delta}(p_1, x) \cup \ldots \cup \hat{\delta}(p_n, x)$$

Language Accepted by a NFA

Definition: The *language* accepted by the NFA $N = (Q, \Sigma, \delta, q_0, F)$ is the set $\mathcal{L}(N) = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) \cap F \neq \emptyset\}$.

That is, a word x is accepted if $\hat{\delta}(q_0, x)$ contains at least one accepting state.

Note: Again, we could write a program that simulates a NFA and let it tell us whether a certain string is accepted or not.

Exercise: Do it!

Transforming a NFA into a DFA

We have seen that for same examples it is much simpler to define a NFA than a DFA.

For example, the language with words of length divisible by 3 or by 5.

However, any language accepted by a NFA is also accepted by a DFA.

In general, the number of states of the DFA is about the number of states in the NFA although it often has many more transitions.

In the worst case, if the NFA has n states, a DFA accepting the same language might have 2^n states.

The *algorithm* transforming a NFA into an equivalent DFA is called the *subset construction*.

The Subset Construction

Definition: Given a NFA $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$ we will construct a DFA $D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$ such that $\mathcal{L}(D) = \mathcal{L}(N)$ as follows:

- $Q_D = \mathcal{P}ow(Q_N)$;
- $\delta_D : Q_D \times \Sigma \to Q_D$ (that is, $\delta_D : \mathcal{P}ow(Q_N) \times \Sigma \to \mathcal{P}ow(Q_N)$)
  $\delta_D(X, a) = \bigcup_{q \in X} \delta_N(q, a)$;
- $F_D = \{S \subseteq Q_N \mid S \cap F_N \neq \emptyset\}$.

Remarks: Subset Construction

- If $|Q_N| = n$ then $|Q_D| = 2^n$.
  If some of the states in Q_D are not *accessible* from the start state of D we can safely remove them (we will see how to do this later on in the course).
- If $X = \{q_1, \ldots, q_n\}$ then $\delta_D(X, a) = \delta_N(q_1, a) \cup \ldots \cup \delta_N(q_n, a)$.

  In addition,

  $$\delta_D(\emptyset, a) = \emptyset \qquad \delta_D(\{q\}, a) = \delta_N(q, a) \qquad \delta_D(X, a) = \bigcup_{q \in X} \delta_D(\{q\}, a)$$

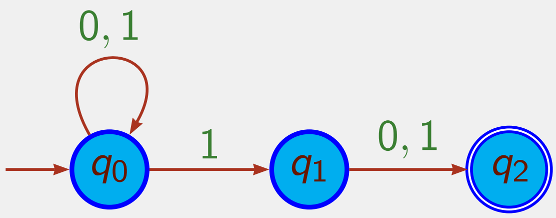
  and

  $$\delta_D(X_1 \cup X_2, a) = \delta_D(X_1, a) \cup \delta_D(X_2, a)$$

- Each accepting state (set) S in F_D contains at least one accepting state of N.

Example: Subset Construction

Let us convert this NFA into a DFA



The DFA we construct will start from $\{q_0\}$. Only accessible states matter.

From $\{q_0\}$, with 0, we can only go to q_0 so $\delta_D(\{q_0\}, 0) = \{q_0\}$.

From $\{q_0\}$, with 1, we can go to q_0 or to q_1. Then, $\delta_D(\{q_0\}, 1) = \{q_0, q_1\}$.

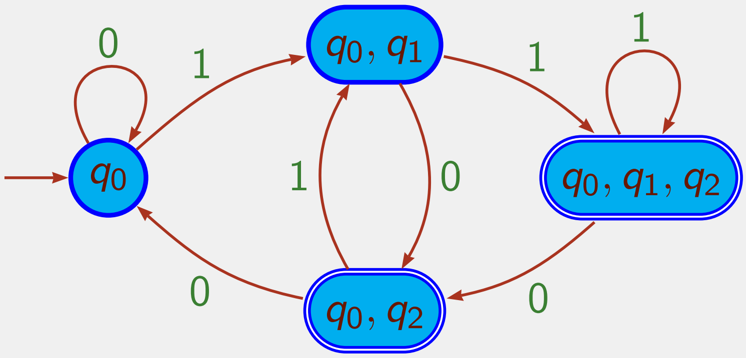
From $\{q_0, q_1\}$, with 0, we can go to q_0 or to q_2. Then, $\delta_D(\{q_0, q_1\}, 0) = \{q_0, q_2\}$.

From $\{q_0, q_1\}$, with 1, we can go to q_0 or q_1 or q_2. Then, $\delta_D(\{q_0, q_1\}, 1) = \{q_0, q_1, q_2\}$.

etc...

Example: Subset Construction (cont.)

The complete (and simplified) DFA from the previous NFA is:



The DFA *remembers* the last two bits seen and accepts a word if the next-to-last bit is 1.

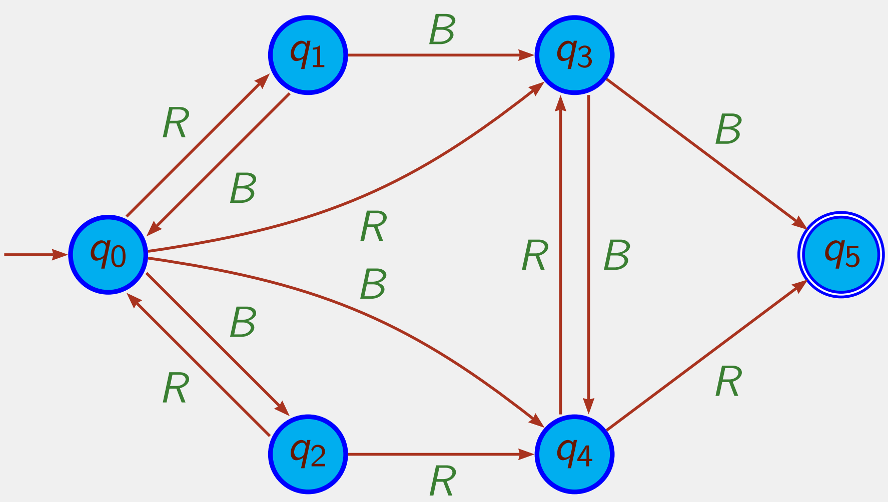
By only computing the *accessible* states (from the start state) we are able to keep the total number of states to 4 (and not 8).

Exercise: Implement the subset construction!

Example: NFA Representation of Gilbreath's Principle

Let us shuffle 2 non-empty alternating decks of cards, one starting with a red card and one starting with a black one.
How does the resulting deck look like?

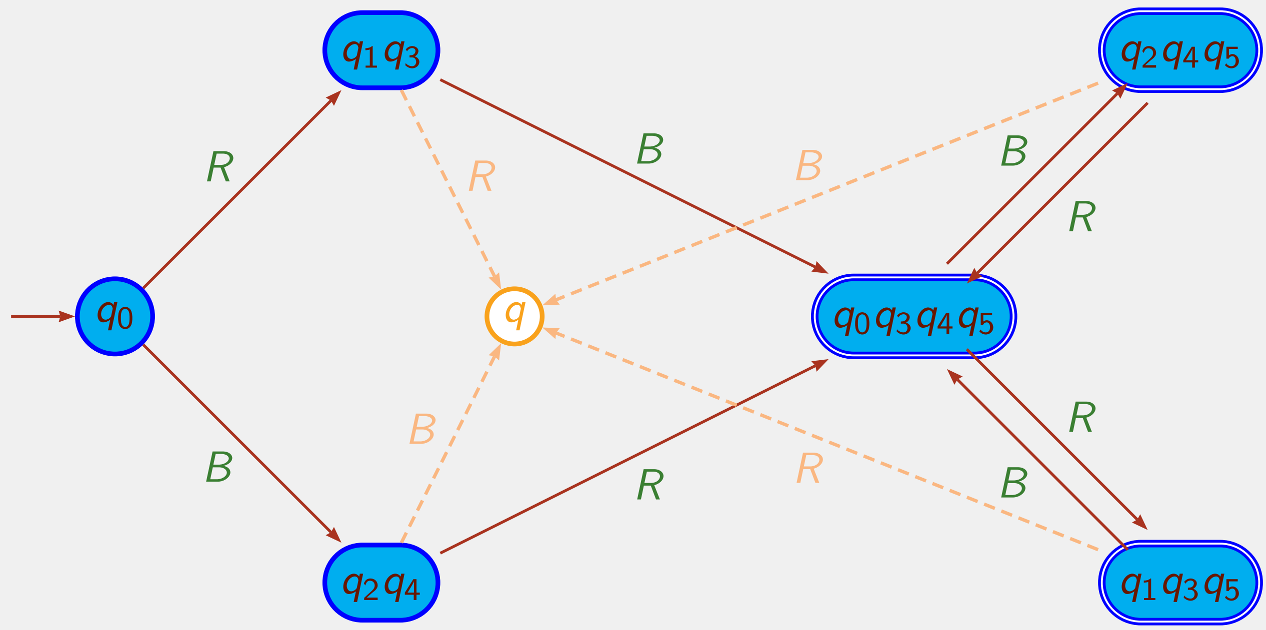
Let $\Sigma = \{B, R\}$ represent a black or red card respectively.



q_0 starts with B and R
q_1 both start with B
q_2 both start with R
q_3 starts with B and ϵ
q_4 starts with R and ϵ
q_5 both ϵ

What does the principle say? Let us build the corresponding DFA!

Example: DFA Representation of Gilbreath's Principle



What does the principle say?

Application of Subset Construction: Text Search

Suppose we want to find occurrences of certain *keywords* in a text.

We could design a NFA that enters in an accepting state when it has recognised one of these keywords.

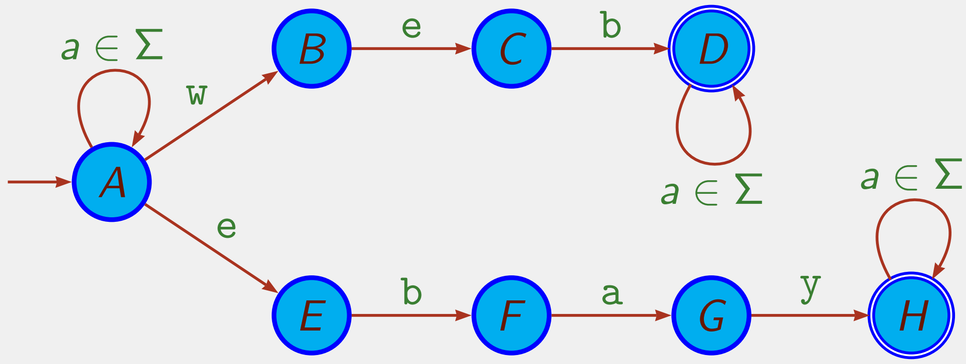
Then we could either implement the NFA or transform it to a DFA and get a deterministic (efficient) program.

We will prove the subset construction correct, then we know the DFA will be correct (if the NFA is!).

This is a good example of a derivation of a *program* (the DFA) from a *specification* (the NFA).

Application of Subset Construction: Text Search

The following (easy to write) NFA searches for the keywords `web` and `ebay`:



If one applies the subset construction one obtains the DFA of page 71 in the book.

Observe that the obtained DFA has the same number of states as the NFA, but it is much more difficult to define directly!

Towards the Correction of the Subset Construction

Proposition: $\forall x. \forall q.\ \hat{\delta}_N(q, x) = \hat{\delta}_D(\{q\}, x)$.

Proof: By induction on x.

Base case is trivial.

The inductive step is:

$$\begin{array}{rcll} \hat{\delta}_N(q, ax) & = & \bigcup_{p \in \delta_N(q,a)} \hat{\delta}_N(p, x) & \text{by definition of } \hat{\delta}_N \\ & = & \bigcup_{p \in \delta_N(q,a)} \hat{\delta}_D(\{p\}, x) & \text{by IH with state } p \\ & = & \hat{\delta}_D(\delta_N(q, a), x) & \text{see lemma below} \\ & = & \hat{\delta}_D(\delta_D(\{q\}, a), x) & \text{remark on slide 10} \\ & = & \hat{\delta}_D(\{q\}, ax) & \text{by definition of } \hat{\delta}_D \end{array}$$

Lemma: *For all words x and sets of states S,* $\hat{\delta}_D(S, x) = \bigcup_{p \in S} \hat{\delta}_D(\{p\}, x)$.

Correction of the Subset Construction

Theorem: *Given a NFA N, if D is the DFA constructed from N by the subset construction then $\mathcal{L}(N) = \mathcal{L}(D)$.*

Proof: $x \in \mathcal{L}(N)$ iff $\hat{\delta}_N(q_0, x) \cap F_N \neq \emptyset$ iff $\hat{\delta}_N(q_0, x) \in F_D$.

By the previous proposition, this is equivalent to $\hat{\delta}_D(\{q_0\}, x) \in F_D$.

Since $\{q_0\}$ is the starting state in D the above is equivalent to $x \in \mathcal{L}(D)$.

Equivalence between DFA and NFA

Theorem: *A language $\mathcal{L}$ is accepted by some DFA iff $\mathcal{L}$ is accepted by some NFA.*

Proof: The "if" part is the result of the previous theorem (correctness of subset construction).

For the "only if" part we need to transform the DFA into a NFA.

Intuitively, each DFA can be seen as a NFA where there exists only one choice at each stage.

Formally, given $D = (Q, \Sigma, \delta_D, q_0, F)$ we define $N = (Q, \Sigma, \delta_N, q_0, F)$ such that $\delta_N(q, a) = \{\delta_D(q, a)\}$.

It only remains to show (by induction on x) that if $\hat{\delta}_D(q_0, x) = p$ then $\hat{\delta}_N(q_0, x) = \{p\}$.

Regular Languages

Recall: A language $\mathcal{L} \subseteq \Sigma^*$ is *regular* iff there exists a DFA D on the alphabet Σ such that $\mathcal{L} = \mathcal{L}(D)$.

Proposition: *A language* $\mathcal{L} \subseteq \Sigma^*$ *is regular iff there exists a NFA* N *such that* $\mathcal{L} = \mathcal{L}(N)$.

Proof: If $\mathcal{L}$ is regular then $\mathcal{L} = \mathcal{L}(D)$ for some DFA D. To any DFA D we can associate a NFA N_D such that $\mathcal{L}(D) = \mathcal{L}(N_D)$ as in previous theorem.

In the other direction, if $\mathcal{L} = \mathcal{L}(N)$ for some NFA N then, the subset construction gives a DFA D such that $\mathcal{L}(N) = \mathcal{L}(D)$ so $\mathcal{L}$ is regular.

Overview of Next Lecture

OBS: Next lecture on Tuesday at *10:00*!

Sections 2.3.6–2.5.5:

- More on NFA;
- NFA with ϵ-transitions.