Finite Automata and Formal Languages
TMV027/DIT321– LP4 2013

Lecture 4
Ana Bove
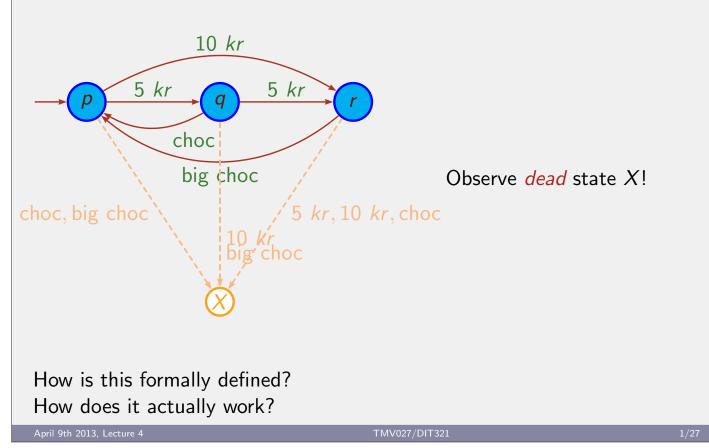
April 9th 2013

Overview of today's lecture:

- Deterministic Finite Automata.

Deterministic Finite Automata

We have already seen examples of DFA:

Observe *dead* state X!

How is this formally defined?
How does it actually work?

Deterministic Finite Automata

Definition: A *deterministic finite automaton* (DFA) 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* $\delta : Q \times \Sigma \to Q$
   (<u>total</u> function that takes as argument a state and a symbol and returns a state);
4. A *start state* $q_0 \in Q$;
5. A set $F \subseteq Q$ of *final* or *accepting* states.

Example: DFA

Let the DFA $(Q, \Sigma, \delta, q_0, F)$ be given by:

$$Q = \{q_0, q_1, q_2\}$$
$$\Sigma = \{0, 1\}$$
$$F = \{q_2\}$$
$$\delta : Q \times \Sigma \to Q$$
$$\delta(q_0, 0) = q_1 \qquad \delta(q_1, 0) = q_1 \qquad \delta(q_2, 0) = q_2$$
$$\delta(q_0, 1) = q_0 \qquad \delta(q_1, 1) = q_2 \qquad \delta(q_2, 1) = q_2$$

What does it do?

How to Represent a DFA?

Transition Diagram: Helps to understand how it works.

Transition Table:

δ	0	1
$\rightarrow q_0$	q_1	q_0
q_1	q_1	q_2
$*q_2$	q_2	q_2

The start state is indicated with $\rightarrow$.
The final states are indicated with $*$.
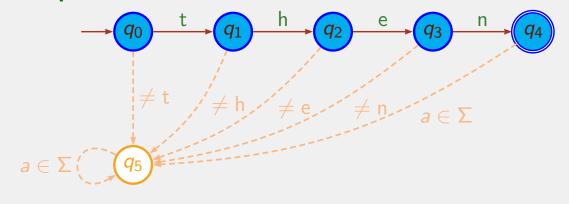
When Does a DFA Accept a Word?

When reading the word the automaton moves according to δ.

Definition: If after reading the input it stops in a final state, it *accepts* the word.

Example:
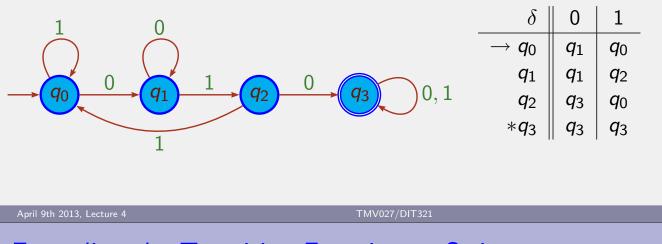

Only the word "then" is accepted.

We have a (non-accepting) *stop* or *dead* state q_5.

Example: DFA

Let us build an automaton that accepts the words that contain 010 as a subword.

That is, given $\Sigma = \{0, 1\}$ we want to accept words in $\mathcal{L} = \{x010y \mid x, y \in \Sigma^*\}$.

Solution: $(\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \delta, q_0, \{q_3\})$ given by

δ	0	1
$\rightarrow q_0$	q_1	q_0
q_1	q_1	q_2
q_2	q_3	q_0
$*q_3$	q_3	q_3

Extending the Transition Function to Strings

How can we compute/determine what happens when we read a certain word?

Definition: We extend δ to strings as $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$.

We define $\hat{\delta}(q, x)$ by recursion on x.

$$\hat{\delta}(q, \epsilon) = q$$
$$\hat{\delta}(q, ax) = \hat{\delta}(\delta(q, a), x)$$

Note: $\hat{\delta}(q, a) = \delta(q, a)$ since the string $a = a\epsilon$.

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

Example: In the previous example, what are $\hat{\delta}(q_0, 10101)$ and $\hat{\delta}(q_0, 00110)$?

Some Properties

Proposition: *For any words x and y, and for any state q we have that $\hat{\delta}(q, xy) = \hat{\delta}(\hat{\delta}(q, x), y)$.*

Proof: We prove $P(x) = \forall q\, y.\hat{\delta}(q, xy) = \hat{\delta}(\hat{\delta}(q, x), y)$ by induction on x.

Base case: $\forall q\, y.\hat{\delta}(q, \epsilon y) = \hat{\delta}(q, y) = \hat{\delta}(\hat{\delta}(q, \epsilon), y)$.

Inductive step: Our IH is that $\forall q\, y.\hat{\delta}(q, xy) = \hat{\delta}(\hat{\delta}(q, x), y)$. We should prove that $\forall q\, y.\hat{\delta}(q, (ax)y) = \hat{\delta}(\hat{\delta}(q, ax), y)$.

$$
\begin{aligned}
\hat{\delta}(q, (ax)y) &= \hat{\delta}(q, a(xy)) && \text{by def of concat} \\
&= \hat{\delta}(\delta(q, a), xy) && \text{by def of } \hat{\delta} \\
&= \hat{\delta}(\hat{\delta}(\delta(q, a), x), y) && \text{by IH with state } \delta(q, a) \\
&= \hat{\delta}(\hat{\delta}(q, ax), y) && \text{by def of } \hat{\delta}
\end{aligned}
$$

Another Definition of $\hat{\delta}$

Recall that we have 2 descriptions of words: $a(b(cd)) = ((ab)c)d$.

We can define $\hat{\delta}'$ as follows:
$$
\begin{aligned}
\hat{\delta}'(q, \epsilon) &= q \\
\hat{\delta}'(q, xa) &= \delta(\hat{\delta}'(q, x), a)
\end{aligned}
$$

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

Proof: By induction on x.

Observe that xa is a special case of xy where $y = a$.

Base case is trivial.

The inductive step goes as follows:
$$
\begin{aligned}
\hat{\delta}(q, xa) &= \hat{\delta}(\hat{\delta}(q, x), a) && \text{by previous prop} \\
&= \delta(\hat{\delta}(q, x), a) && \text{by def of } \hat{\delta} \\
&= \delta(\hat{\delta}'(q, x), a) && \text{by IH} \\
&= \hat{\delta}'(q, xa) && \text{by def of } \hat{\delta}'
\end{aligned}
$$

Language Accepted by a DFA

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

Example: In the example on slide 6, 10101 is accepted but 00110 is not.

Note: We could write a program that simulates a DFA and let the program tell us whether a certain string is accepted or not.

Functional Representation of a DFA Accepting $x010y$

```
data Q = Q0 | Q1 | Q2 | Q3
data S = O | I

final :: Q -> Bool
final Q3 = True
final _  = False

delta :: Q -> S -> Q
delta Q0 O = Q1
delta Q0 I = Q0
delta Q1 O = Q1
delta Q1 I = Q2
delta Q2 O = Q3
delta Q2 I = Q0
delta Q3 _ = Q3
```

Functional Representation of a DFA Accepting $x010y$

```
run :: Q -> [S] -> Q
run q [] = q
run q (a:xs) = run (delta q a) xs


accepts :: [S] -> Bool
accepts xs = final (run Q0 xs)
```
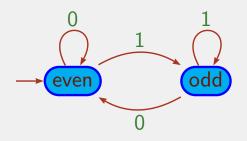
Accepting by End of String

Sometimes we use an automaton to identify properties of a certain string.

Here, the important things is the state the automaton is in when we finish reading the input.
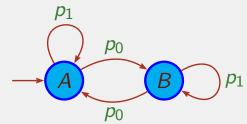
Then, the set of final states is actually not needed and can be omitted.

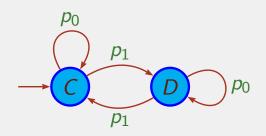**Example:** The following automaton determines whether a binary number is even or odd.

Product of Automata

Given an automaton that determines whether the number of p_0's is even or odd

State A: even number of p_0's
State B: odd number of p_0's

and an automaton that determines whether the number of p_1's is even or odd

State C: even number of p_1's
State D: odd number of p_1's

How can we combine them to keep track of the parity of *both* p_0 and p_1?
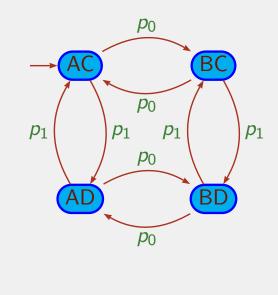
Product Construction

Definition: Given two DFA $D_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $D_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ with the *same alphabet* Σ, we can define the *product* $D = D_1 \times D_2$ as follows:

- $Q = Q_1 \times Q_2$;
- $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$;
- $q_0 = (q_1, q_2)$;
- $F = F_1 \times F_2$.

Proposition: $\hat{\delta}((r_1, r_2), x) = (\hat{\delta}_1(r_1, x), \hat{\delta}_2(r_2, x))$.

Proof: By induction on x.

Example: Product of Automata (from slide 14)

The product automaton that keeps track of the parity of *both* p_0 and p_1 is:

State AC: even nr. of p_0's and p_1's

State BC: odd nr. of p_0's and
even nr. of p_1's

State AD: even nr. of p_0's and
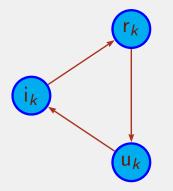odd nr. of p_1's

State BD: odd nr. of p_0's and p_1's

Example: Product of Automata

Consider a system where users have three states: *idle*, *requesting* and *using*.
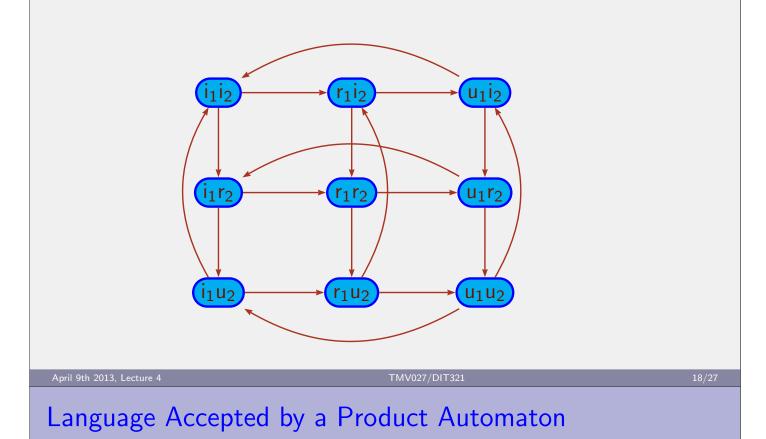
Let us assume we have 2 users.

Each user is represented by a simple automaton, for $k = 1, 2$:

Example: Product of Automata (cont.)

The complete system is represented by the product of these 2 automata and it has 3 * 3 = 9 states.

Language Accepted by a Product Automaton

Proposition: *Given two DFA D_1 and D_2, then*
$\mathcal{L}(D_1 \times D_2) = \mathcal{L}(D_1) \cap \mathcal{L}(D_2)$.

Proof: $\hat{\delta}(q_0, x) = (\hat{\delta}_1(q_1, x), \hat{\delta}_2(q_2, x)) \in F$ iff $\hat{\delta}_1(q_1, x) \in F_1$ and $\hat{\delta}_2(q_2, x) \in F_2$, that is, $x \in \mathcal{L}(D_1)$ and $x \in \mathcal{L}(D_2)$ iff $x \in \mathcal{L}(D_1) \cap \mathcal{L}(D_2)$.

Note: It can be quite difficult to directly build an automaton accepting the intersection of two languages.

Exercise: Build a DFA for the language that contains the subword *abb* twice and an even number of *a*'s.

Variation of the Product

Definition: We define $D_1 \oplus D_2$ similarly to $D_1 \times D_2$ but with a different notion of accepting state:

$$\text{a state } (r_1, r_2) \text{ is accepting iff } r_1 \in F_1 \text{ } or \text{ } r_2 \in F_2$$

Proposition: *Given two DFA D_1 and D_2, then*
$\mathcal{L}(D_1 \oplus D_2) = \mathcal{L}(D_1) \cup \mathcal{L}(D_2)$.

Exercise: Define an automaton accepting strings with lengths multiple of 3 or of 5.
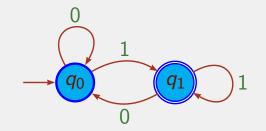
Complement

Definition: Given the automaton $D = (Q, \Sigma, \delta, q_0, F)$ we define the *complement* $\overline{D}$ of D as the automaton $\overline{D} = (Q, \Sigma, \delta, q_0, Q - F)$.
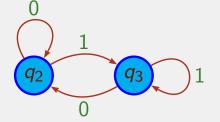
Proposition: *Given a DFA D we have that $\mathcal{L}(\overline{D}) = \Sigma^* - \mathcal{L}(D) = \overline{\mathcal{L}(D)}$.*

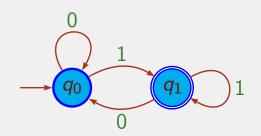**Note:** We have that $D_1 \oplus D_2 = \overline{\overline{D_1} \times \overline{D_2}}$.

Accessible Part of a DFA

Consider the DFA $(\{q_0, \ldots, q_3\}, \{0, 1\}, \delta, q_0, \{q_1\})$ given by

This is clearly equivalent to the DFA

which is the *accessible* part of the DFA. The states q_2 and q_3 are not accessible/reachable from the start state and can be removed.

Accessible States

Definition: The set $\mathrm{Acc} = \{\hat{\delta}(q_0, x) \mid x \in \Sigma^*\}$ is the set of *accessible* states (from the state q_0).

Proposition: *If $D = (Q, \Sigma, \delta, q_0, F)$ is a DFA, then $D' = (Q \cap \mathrm{Acc}, \Sigma, \delta|_{Q \cap \mathrm{Acc}}, q_0, F \cap \mathrm{Acc})$ is a DFA such that $\mathcal{L}(D) = \mathcal{L}(D')$.*

Proof: Notice that D' is well defined and that $\mathcal{L}(D') \subseteq \mathcal{L}(D)$.

If $x \in \mathcal{L}(D)$ then $\hat{\delta}(q_0, x) \in F$. By definition $\hat{\delta}(q_0, x) \in \mathrm{Acc}$. Hence $\hat{\delta}(q_0, x) \in F \cap \mathrm{Acc}$ and then $x \in \mathcal{L}(D')$.

Regular Languages

Recall: Given an alphabet Σ, a *language* $\mathcal{L}$ is a subset of Σ^*, that is, $\mathcal{L} \subseteq \Sigma^*$.

Definition: 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: *If $\mathcal{L}_1$ and $\mathcal{L}_2$ are regular languages then so are $\mathcal{L}_1 \cap \mathcal{L}_2$, $\mathcal{L}_1 \cup \mathcal{L}_2$ and $\Sigma^* - \mathcal{L}_1$.*

Proof: ...

Application of DFA: Automatic Theorem Proving

Assume $\Sigma = \{a, b\}$.

Let $\mathcal{L}$ be the set of $x \in \Sigma^*$ such that any a in x is followed by a b.

Let $\mathcal{L}'$ be the set of $x \in \Sigma^*$ such that any b in x is followed by a a.

How to prove that $\mathcal{L} \cap \mathcal{L}' = \{\epsilon\}$?

Intuitively:

- if $x \neq \epsilon$ in $\mathcal{L}$ we have that if $x = \ldots a \ldots$ then it should actually be $x = \ldots a \ldots b \ldots$
- if $x \neq \epsilon$ in $\mathcal{L}'$ we have that if $x = \ldots b \ldots$ then it should actually be $x = \ldots b \ldots a \ldots$

Hence a non-empty word in $\mathcal{L} \cap \mathcal{L}'$ should be infinite.

Application of DFA: Automatic Theorem Proving (cont.)

Formally we can automatically prove that $\mathcal{L} \cap \mathcal{L}' = \{\epsilon\}$ with an automaton.

Define a DFA D such that $\mathcal{L}(D) = \mathcal{L}$.

Define a DFA D' such that $\mathcal{L}(D') = \mathcal{L}'$.

Now we can compute $D \times D'$ and check that

$$\mathcal{L} \cap \mathcal{L}' = \mathcal{L}(D \times D') = \{\epsilon\}$$

Overview of Next Lecture

Sections 2.3–2.3.5, brief on 2.4:

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