Finite Automata and Formal Languages

TMV026/DIT321– LP4 2011

Ana Bove

Lecture 3

March 24th 2011

Overview of today's lecture:

- **Central Concepts of Automata Theory**
- **Deterministic Finite Automata**

Some Concepts in Set Theory

Empty set: $\emptyset$ is the set with no elements. We have that $\emptyset \subseteq S$ for all sets S.

Singleton sets: Sets with only one element: $\{p_0\}$, $\{p_1\}$

Finite sets: Set with a finite number n of elements:
$\{p_1, \ldots, p_n\} = \{p_1\} \bigcup \cdots \bigcup \{p_n\}$

Union: $S_1 \cup S_2 = \{x \mid x \in S_1 \text{ or } x \in S_2\}$

Intersection: $S_1 \cap S_2 = \{x \mid x \in S_1 \text{ and } x \in S_2\}$

Complement: $S - A$ is the set of all elements in set S not in set A.
When the set S is known, $S - A$ is sometimes written $\overline{A}$.

Power sets: $\mathcal{P}ow(S)$ the set of all subsets of the set S.
Observe that $\emptyset \in \mathcal{P}ow(S)$.

Central Concepts of Automata Theory

Alphabets

Definition: An *alphabet* is a finite non-empty set of symbols, usually denoted by Σ.

The number of symbols in Σ is denoted as $|\Sigma|$.

Note: Alphabets will represent the observable events of the automata.

Example: Some alphabets:

- on/off-switch $\Sigma = \{\mathsf{Push}\}$
- simple vending machine $\Sigma = \{5\ kr, \mathsf{choc}\}$
- complex vending machine $\Sigma = \{5\ kr, 10\ kr, \mathsf{choc}, \mathsf{big\ choc}\}$
- parity counter $\Sigma = \{p_0, p_1\}$

Type convention: We will use $a, b, c, \ldots$ to denote symbols.

Strings or Words

Definition: *Strings/Words* are finite sequence of symbols from some alphabet.

A word will represent the *behaviour* of an automaton.

Example: Some behaviours:

- on/off-switch $\mathsf{Push\ Push\ Push\ Push}\ \ldots$
- simple vending machine $5\ kr\ \mathsf{choc}\ 5\ kr\ \mathsf{choc}\ 5\ kr\ \mathsf{choc}\ \ldots$
- parity counter p_0p_1 or $p_0p_0p_0p_1p_1p_0$ or $\ldots$

Type convention: We will use $w, x, y, z, \ldots$ to denote words.

Inductive Definition of Σ^*

Definition: Σ^* is the set of all words for a given alphabet Σ.

This can be described inductively in at least 2 different ways:

1. Basis case: the empty word ϵ is in Σ^* (notation: $\epsilon \in \Sigma^*$)
   Inductive step: if $a \in \Sigma$ and $x \in \Sigma^*$ then $ax \in \Sigma^*$

2. Basis case: $\epsilon \in \Sigma^*$
   Inductive step: if $a \in \Sigma$ and $x \in \Sigma^*$ then $xa \in \Sigma^*$

We can (recursively) *define* functions over Σ^* and (inductively) *prove* properties about those functions.

Length

Definition: The *length* function $|\_| : \Sigma^* \to \mathbb{N}$ is defined as:

$$|\epsilon| = 0$$
$$|ax| = 1 + |x|$$

Example: $|p_0p_1p_1p_0p_0| = 5$

Concatenation

Definition: Given the strings x and y, the *concatenation* xy is defined as:

$$\epsilon y = y$$
$$(ax)y = a(xy)$$

Example: Observe that in general $xy \neq yx$.

If $x = p_0p_1p_1$ and $y = p_0p_0$ then $xy = p_0p_1p_1p_0p_0$ and $yx = p_0p_0p_0p_1p_1$.

Lemma: *If Σ has more than one symbol then concatenation is not commutative.*

Power

Of a string: We define x^n as follows:

$$x^0 = \epsilon$$
$$x^{n+1} = xx^n$$

Example: $(p_0p_1p_0)^3 = p_0p_1p_0p_0p_1p_0p_0p_1p_0$

Of an alphabet: We define Σ^n, the set of words over Σ with length n, as follows:

$$\Sigma^0 = \{\epsilon\}$$
$$\Sigma^{n+1} = \{ax \mid a \in \Sigma,\ x \in \Sigma^n\}$$

Example: $\{0,1\}^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$.

Observe: $\Sigma^* = \Sigma^0 \bigcup \Sigma^1 \bigcup \Sigma^2 \ldots$ and $\Sigma^+ = \Sigma^1 \bigcup \Sigma^2 \bigcup \Sigma^3 \ldots$

Reverse Function

Intuitively, $\mathsf{rev}(a_1 \dots a_n) = a_n \dots a_1$.

Definition: Formally we can define $\mathsf{rev}(x)$ as:

$$\mathsf{rev}(\epsilon) = \epsilon$$
$$\mathsf{rev}(ax) = \mathsf{rev}(x)a$$

Some Properties

Lemma: *Concatenation is associative:* $\forall x, y, z.\ x(yz) = (xy)z$.

We shall simply write xyz.

Lemma: $\forall x, y.\ |xy| = |x| + |y|$.

Lemma: $\forall x, y.\ x\epsilon = \epsilon x = x$.

Lemma: $\forall x.\ |x^n| = n|x|$.

Lemma: $\forall \Sigma.\ |\Sigma^n| = |\Sigma|^n$.

Lemma: $\forall x, \mathsf{rev}(\mathsf{rev}(x)) = x$.

Lemma: $\forall x, y.\ \mathsf{rev}(xy) = \mathsf{rev}(y)\mathsf{rev}(x)$.

All these properties can be proved by induction.

Some Terminology

Definition: Given x and y words over a certain alphabet Σ:

- x is a *prefix* of y iff there exists z such that $y = xz$
- x is a *suffix* of y iff there exists z such that $y = zx$
- x is a *palindrome* iff $x = \mathsf{rev}(x)$

Languages

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

Note: If $\mathcal{L} \subseteq \Sigma^*$ and $\Sigma \subseteq \Delta$ then $\mathcal{L} \subseteq \Delta^*$.

Note: A language can be either finite or infinite.

Example: Some languages:

- Swedish, English, Spanish, French, ...
- Any programming language
- $\emptyset$, $\{\epsilon\}$ and Σ^* are languages over any Σ
- The set of prime natural numbers $\{1, 3, 5, 7, 11, \ldots\}$

Some Operations on Languages

Definition: Given $\mathcal{L}$, $\mathcal{L}_1$ and $\mathcal{L}_2$ languages, we define the following languages:

Union, Intersection, ... : As for any set

Concatenation: $\mathcal{L}_1\mathcal{L}_2 = \{x_1x_2 \mid x_1 \in \mathcal{L}_1,\ x_2 \in \mathcal{L}_2\}$

Closure: $\mathcal{L}^* = \bigcup_{n\in\mathbb{N}} \mathcal{L}^n$
where $\mathcal{L}^0 = \{\epsilon\}$, $\mathcal{L}^{n+1} = \mathcal{L}^n\mathcal{L}$.

Note: We have then that $\emptyset^* = \{\epsilon\}$ and
$\mathcal{L}^* = \mathcal{L}^0 \cup \mathcal{L}^1 \cup \mathcal{L}^2 \cup \ldots = \{\epsilon\} \cup \{x_1 \ldots x_n \mid n > 0, x_i \in \mathcal{L}\}$

Notation: $\mathcal{L}^+ = \mathcal{L}^1 \cup \mathcal{L}^2 \cup \mathcal{L}^3 \cup \ldots$ and $\mathcal{L}? = \mathcal{L} \cup \{\epsilon\}$.

How to Prove the Equality of Languages?

Given the languages $\mathcal{L}$ and $\mathcal{M}$, how can we prove that $\mathcal{L} = \mathcal{M}$?

A few possibilities:

- Languages are sets so we prove that $\mathcal{L} \subseteq \mathcal{M}$ and $\mathcal{M} \subseteq \mathcal{L}$
- We can reason about the elements in the language:

  Example: $\{a(ba)^n \mid n \geqslant 0\} = \{(ab)^n a \mid n \geqslant 0\}$ can be proved by induction on n.

- Transitivity of equality: $\mathcal{L} = \mathcal{L}_1 = \ldots = \mathcal{L}_m = \mathcal{M}$

Algebraic Laws for Languages

The following equalities hold for any languages $\mathcal{L}$, $\mathcal{M}$ and $\mathcal{N}$.

- Associativity: $\mathcal{L} \cup (\mathcal{M} \cup \mathcal{N}) = (\mathcal{L} \cup \mathcal{M}) \cup \mathcal{N}$.
  $\mathcal{L} \cap (\mathcal{M} \cap \mathcal{N}) = (\mathcal{L} \cap \mathcal{M}) \cap \mathcal{N}$ and $\mathcal{L}(\mathcal{M}\mathcal{N}) = (\mathcal{L}\mathcal{M})\mathcal{N}$
- Commutative: $\mathcal{L} \cup \mathcal{M} = \mathcal{M} \cup \mathcal{L}$ and $\mathcal{L} \cap \mathcal{M} = \mathcal{M} \cap \mathcal{L}$
- In general, concatenation is not commutative: $\mathcal{L}\mathcal{M} \neq \mathcal{M}\mathcal{L}$
- Distributivity: $\mathcal{L}(\mathcal{M} \cup \mathcal{N}) = \mathcal{L}\mathcal{M} \cup \mathcal{L}\mathcal{N}$ and $(\mathcal{M} \cup \mathcal{N})\mathcal{L} = \mathcal{M}\mathcal{L} \cup \mathcal{N}\mathcal{L}$
- Identity (or neutral): $\mathcal{L} \cup \emptyset = \emptyset \cup \mathcal{L} = \mathcal{L}$ and $\mathcal{L}\{\epsilon\} = \{\epsilon\}\mathcal{L} = \mathcal{L}$
- Annihilator: $\mathcal{L}\emptyset = \emptyset\mathcal{L} = \emptyset$
- Idempotent: $\mathcal{L} \cup \mathcal{L} = \mathcal{L}$,$\mathcal{L} \cap \mathcal{L} = \mathcal{L}$
- $\emptyset^* = \{\epsilon\}^* = \{\epsilon\}$
- $\mathcal{L}^+ = \mathcal{L}\mathcal{L}^* = \mathcal{L}^*\mathcal{L}$
- $(\mathcal{L}^*)^* = \mathcal{L}^*$

Algebraic Laws for Languages (Cont.)

Note: While

$$\mathcal{L}(\mathcal{M} \cap \mathcal{N}) \subseteq \mathcal{L}\mathcal{M} \cap \mathcal{L}\mathcal{N} \quad \text{and} \quad (\mathcal{M} \cap \mathcal{N})\mathcal{L} \subseteq \mathcal{M}\mathcal{L} \cap \mathcal{N}\mathcal{L}$$

both hold, in general

$$\mathcal{L}\mathcal{M} \cap \mathcal{L}\mathcal{N} \subseteq \mathcal{L}(\mathcal{M} \cap \mathcal{N}) \quad \text{and} \quad \mathcal{M}\mathcal{L} \cap \mathcal{N}\mathcal{L} \subseteq (\mathcal{M} \cap \mathcal{N})\mathcal{L}$$

don't.

Consider for example the case where

$$\mathcal{L} = \{\epsilon, a\}, \quad \mathcal{M} = \{a\}, \quad \mathcal{N} = \{aa\}$$

Then $\mathcal{L}\mathcal{M} \cap \mathcal{L}\mathcal{N} = \{aa\}$ but $\mathcal{L}(\mathcal{M} \cap \mathcal{N}) = \mathcal{L}\emptyset = \emptyset$.

Functions between Languages

Definition: A function $f : \Sigma^* \to \Delta^*$ between 2 languages should be such that it satisfies

$$f(\epsilon) = \epsilon$$
$$f(xy) = f(x)f(y)$$

Intuitively, $f(a_1 \dots a_n) = f(a_1) \dots f(a_n)$.

Notice that $f(a) \in \Delta^*$ if $a \in \Sigma$.

Definition: f is called *coding* iff f is *injective*.

Definition: $f(\mathcal{L}) = \{f(x) \mid x \in \mathcal{L}\}$.

Some Terminology

Definition: A *problem* is the question of deciding if a given string is a member of some particular language.

A "problem" can be expressed as membership in a language.

If $\mathcal{L}$ is a language over Σ then the problem $\mathcal{L}$ is:

given $w \in \Sigma^*$ decide whether or not w is in $\mathcal{L}$.

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* $\delta : Q \times \Sigma \to Q$
   (total 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

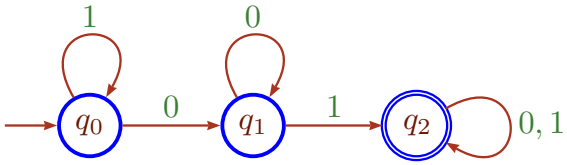
Let the DFA $(Q, \Sigma, \delta, q_0, F)$ be given by:

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

What does it do?

How to Represent a DFA?

Transition Diagram: As we have seen before.



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 $\rightarrow$.

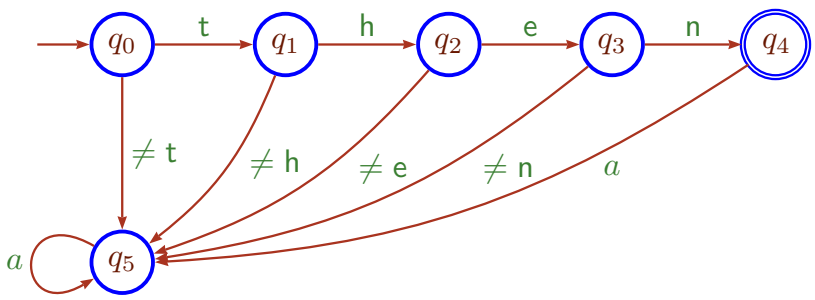
The final states are indicated with $*$.

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.

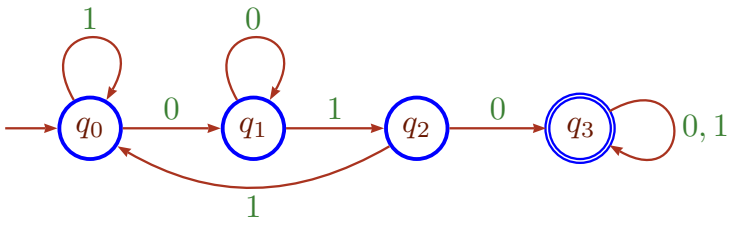
We have a (non-accepting) *stop* or *dead* state q_5.

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 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 the result by induction on x.

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

Inductive step: Given $\hat{\delta}(q, xy) = \hat{\delta}(\hat{\delta}(q, x), y)$ for any word y and any state q, we should prove that $\hat{\delta}(q, (ax)y) = \hat{\delta}(\hat{\delta}(q, ax), y)$.

$\hat{\delta}(q, (ax)y) = \hat{\delta}(q, a(xy)) = \hat{\delta}(\delta(q, a), xy) = IH = \hat{\delta}(\hat{\delta}(\delta(q, a), x), y) = \hat{\delta}(\hat{\delta}(q, ax), y)$

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:

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

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

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

Basis case is trivial.

The inductive step goes as follows:

$\hat{\delta}(q, xa) = \hat{\delta}(\hat{\delta}(q, x), a) = \delta(\hat{\delta}(q, x), a) = IH = \delta(\hat{\delta}'(q, x), a) = \hat{\delta}'(q, xa)$.

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 previous example, 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)


%% Alternatively, given that
%% run q [x1,...,xn] = delta (... (delta Q0 x1) ...) xn

run :: Q -> [S] -> Q
run = foldl delta

accepts :: [S] -> Bool
accepts = final . run Q0
```

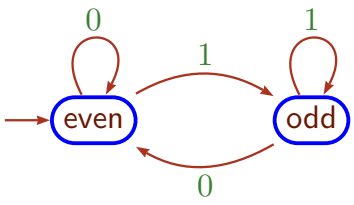

Accepting by End of String

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

In these cases, the important thing is the state the automaton is in when we finish reading the input.

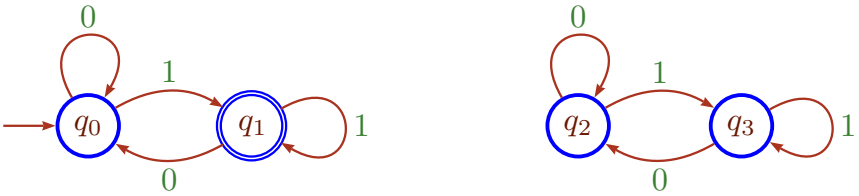
Here, the the set of final states is actually not needed and can be omitted.

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

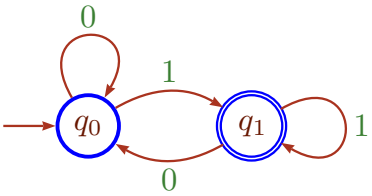


Accessible Part of a DFA

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 from the start state and can be removed.

Accessible States

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

Proposition: *If $D = (Q, \Sigma, \delta, q_0, F)$ is a DFA, then $D' = (Q \cap \mathsf{Acc}, \Sigma, \delta', q_0, F \cap \mathsf{Acc})$, where δ' is the function δ restricted to the states in $Q \cap \mathsf{Acc}$, 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 $\hat{\delta}(q_0, x) \in F$. Observe that by definition $\hat{\delta}(q_0, x) \in \mathsf{Acc}$. Hence $\hat{\delta}(q_0, x) \in F \cap \mathsf{Acc}$ and then $x \in \mathcal{L}(D')$.

Automatic Theorem Proving

Recall the example $f, g, h : \mathbb{N} \to \{0, 1\}$ such that:

$$\begin{array}{lll} f(0) = 0 & g(0) = 1 & h(0) = 0 \\ f(n+1) = g(n) & g(n+1) = f(n) & h(n+1) = 1 - h(n) \end{array}$$

We can prove $\forall n.\ h(n) = f(n)$ automatically using a DFA.

- $Q = \{0, 1\} \times \{0, 1\} \times \{0, 1\}$
- $\Sigma = \{1\}$ (The number n is represented by 1^n and 0 by $1^0 = \epsilon$)
- $q_0 = (f(0), g(0), h(0)) = (0, 1, 0)$.
- $\hat{\delta}((0, 1, 0), 1^n) = (f(n), g(n), h(n))$
  A transition goes from $(f(n), g(n), h(n))$ to $(f(n+1), g(n+1), h(n+1))$ and then $\delta((a, b, c), s) = (b, a, 1 - c)$

We check that all accessible states (a, b, c) satisfy $a = c$, that is, the property $a = c$ is an invariant for each transition of the automata

Automatic Theorem Proving

A more complex example:

$$f(0) = 0 \qquad f(1) = 1 \qquad f(n+2) = f(n) + f(n+1) - 2f(n)f(n+1)$$

We have

$$f(0) = 0 \quad f(1) = 1 \quad f(2) = 1 \quad f(3) = 0 \quad f(4) = 1 \quad f(5) = 1 \quad \ldots$$

Show that $f(n+3) = f(n)$ by using

- $Q = \{0, 1\} \times \{0, 1\} \times \{0, 1\}$
- $\Sigma = \{1\}$
- $q_0 = (f(0), f(1), f(2)) = (0, 1, 1)$
- $\delta((a, , b, c), s) = (b, c, b + c - 2bc)$