Finite Automata and Formal Languages

TMV026/DIT321 – LP4 2010

Lecture 2

March 18th 2010

Overview of today's lecture:

- **Formal Proofs**
- **Central Concepts of Automata Theory**

Example: Proof by Induction

Proposition: *If $n \geqslant 8$ then n can be written as a sum of 3's and 5's.*

Proof: Let $P(n)$ be "n can be written as a sum of 3's and 5's".

$P(7)$ does not hold but $P(8), P(9)$ and $P(10)$ hold.

Now we want to prove that if $P(8), P(9), \ldots, P(n)$ hold for $n \geqslant 10$ then $P(n+1)$ holds.

Observe that if $n \geqslant 10$ then $n + 1 - 3 \geqslant 8$.

Hence $P(n+1-3)$ holds and so must $P(n+1)$.

Example: Proof by Induction

Proposition: *All horses have the same colour.*

Proof: Let $P(n)$ be "in any set of n horses they all have the same colour".

$P(0)$ is not interesting in this example.

$P(1)$ is clearly true.

Let us show that $P(n)$ implies $P(n+1)$.

Let $h_1, h_2, \ldots, h_n, h_{n+1}$ be a set of $n+1$ horses.

Take $h_1, h_2, \ldots, h_n$. By inductive hypothesis (IH) they all have the same colour.

Take now $h_2, h_3, \ldots, h_n, h_{n+1}$. Again, by IH they all have the same colour.

Hence all horses $h_1, h_2, \ldots, h_n$ must have the same colour.

Mutual Induction

Sometimes we cannot prove a single statement $P(n)$ but rather a group of statements $P_1(n), P_2(n), \ldots, P_k(n)$ simultaneously by induction on n.

This is very common in automata theory where we need an statement for each of the states of the automata.

Example: Recall the on/off-switch.

Example: Proof by Mutual Induction

Let $f, g, h : \mathbb{N} \to \{0, 1\}$ be as follows:

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

Proposition: $\forall n.\ h(n) = f(n)$.

Proof: If $P(n)$ is $h(n) = f(n)$ it does not seem possible to prove $P(n) \Rightarrow P(n+1)$ directly.

We strengthen $P(n)$ to $P'(n)$ as follows:

Let $P'(n)$ be $h(n) = f(n) \wedge h(n) = 1 - g(n)$.

We prove $P'(0)$, that is, $h(0) = f(0) \wedge h(0) = 1 - g(0)$.

Then we prove that $P'(n+1)$ follows from $P'(n)$.

Automaton Representation

We can think of f, g and h as a *circuit*.

The circuit can be represented as an automaton as follows:

- The states are the possible values of $s(n) = (f(n), g(n), h(n))$
- The transitions are from the states $s(n)$ to the state $s(n+1)$
- Initial state is $s(0) = (0, 1, 0)$

One can check the invariant $f(n) = h(n)$ on all the states accessible from the initial state.

Inductively Defined Sets

Natural Numbers: Basis case: 0 is a natural number

Inductive step: If n is a natural number then $n + 1$ is a natural number

Finite Lists: Basic case: [] is the empty list over any set A

Inductive step: If $a \in A$ and xs is a list over A then $a : xs$ is a list over A.

Finitely Branching Trees: Basic case: () is a tree over any set A

Inductive step: If $t_1, \ldots, t_k$ are tree over the set A, then $(t_1, \ldots, t_k)$ is a tree over A.

...

We can then define functions on the set of trees by recursion, and prove properties of these functions by *structural induction*.

Inductively Defined Sets

To define a set S we need to specify:

Basic cases: Here we say which specific elements $e_1, \ldots, e_m$ belong to S.

Inductive steps: Assuming that $s_1, \ldots, s_n$ belong to S, we indicate how to use $s_1, \ldots, s_n$ in order to construct new elements of S $\mathsf{c}_1[s_1, \ldots, s_n], \ldots, \mathsf{c}_k[s_1, \ldots, s_n]$.

Example: Simple boolean expressions:

Basic cases: true and false are boolean expressions

Inductive steps: if a and b are boolean expression then

$$(a) \qquad \mathsf{not}\ a \qquad a\ \mathsf{and}\ b \qquad a\ \mathsf{or}\ b$$

are also boolean expressions

Proofs by Structural Induction

Given an inductively defined set S, we prove $\forall s \in S.\ P(s)$ by:

Basic cases: We prove that $P(e_1), \ldots, P(e_m)$ hold.

Inductive steps: Assuming that $P(s_1), \ldots, P(s_n)$ hold, we prove that $P(\mathsf{c_1}[s_1, \ldots, s_n]), \ldots, P(\mathsf{c_k}[s_1, \ldots, s_n])$ also hold.

Example: Proof by Structural Induction

Given the finite lists, let us (recursively) define the append and length functions:

$$[] +\!\!+\ ys = ys \qquad \mathsf{len}\ [] = 0$$
$$(a : xs) +\!\!+\ ys = a : (xs +\!\!+\ ys) \qquad \mathsf{len}\ (a : xs) = 1 + \mathsf{len}\ xs$$

Proposition: $\forall xs, ys.\ \mathsf{len}\ (xs +\!\!+\ ys) = \mathsf{len}\ xs + \mathsf{len}\ ys$.

Proof: By structural induction on xs.

Basic: We prove $P([])$.

Inductive step: We show that $P(xs)$ implies $P(a : xs)$.

Example: Proof by Structural Induction

Given the finitely branching trees, let us (recursively) define functions counting the number of edges and of nodes:

$$\mathsf{ne}() = 0 \qquad \mathsf{nn}() = 1$$
$$\mathsf{ne}(t_1, \ldots, t_k) = k + \qquad \mathsf{nn}(t_1, \ldots, t_k) = 1 +$$
$$\mathsf{ne}(t_1) + \ldots + \mathsf{ne}(t_k) \qquad \mathsf{ne}(t_1) + \ldots + \mathsf{ne}(t_k)$$

Proposition: $\forall t.\ \mathsf{nn}(t) = 1 + \mathsf{ne}(t)$.

Proof: By structural induction on t.

Basic: We prove $P()$.

Inductive step: We show that $P(t_1, \ldots, t_k)$ follows from $P(t_1), \ldots, P(t_k)$.

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:
$\{p_0, \ldots, p_n\} = \{p_0\} \bigcup \ldots \bigcup \{p_n\}$

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

Central Concepts of Automata Theory

Alphabets

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

It will represent the observable events of the automaton.

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 $\mathsf{Push}\ \mathsf{Push}\ \mathsf{Push}\ \mathsf{Push}\ \ldots$
- 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: *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 Functions

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

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 then we define the following languages:

Union: $\mathcal{L}_1 \cup \mathcal{L}_2 = \{x \mid x \in \mathcal{L}_1 \text{ or } x \in \mathcal{L}_2\}$

Intersection: $\mathcal{L}_1 \cap \mathcal{L}_2 = \{x \mid x \in \mathcal{L}_1 \text{ and } x \in \mathcal{L}_2\}$

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: $\mathcal{L} \cap \mathcal{L} = \mathcal{L}$
- $\emptyset^* = \{\epsilon\}^* = \{\epsilon\}$
- $\mathcal{L}^+ = \mathcal{L}\mathcal{L}^* = \mathcal{L}^*\mathcal{L}$
- $(\mathcal{L}^*)^* = \mathcal{L}^*$

Algebraic Laws for Languages

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:

Let the function $f : \Sigma^* \to \Delta^*$ satisfying

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

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

Then, f is called ***coding*** iff f is *injective*.

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 $\mathcal{L}$.