Finite Automata Theory and Formal Languages

TMV027/DIT321– LP4 2017

Lecture 4
Ana Bove

March 24th 2017

Overview of today's lecture:

- Structural induction;
- Concepts of automata theory.

Recap: Formal Proofs

- How formal should a proof be? Depends on its purpose...
- ... but should be convincing ...
  ... and the validity of each step should be easily understood;
- One proves the conclusions assuming the validity of the hypotheses!
- Different kind of proofs (contradiction, contrapositive, counterexample, induction, ...)
- Simple/strong induction allows to prove a property over *all* Natural numbers;
- Sometimes we prove several properties that depend on each other (mutual induction);
- Inductive definitions generate possibly infinite sets with finite elements: Booleans, Natural numbers, lists, trees, ...
- We can recursively defined functions over inductive sets.

Inductively Defined Sets (Recap)

To define a set S by induction we need to specify:

Base cases: $e_1, \dots, e_m \in S$;

Inductive steps: Given $s_1, \dots, s_{n_i} \in S$,
then $\mathsf{c}_1[s_1, \dots, s_{n_1}], \dots, \mathsf{c}_\mathsf{k}[s_1, \dots, s_{n_k}] \in S$;

Closure: There is no other way to construct elements in S.
(We will usually omit this part.)

Proofs by Structural Induction

Generalisation of mathematical induction to other inductively defined sets such as lists, trees, ...

VERY useful in computer science: it allows to prove properties over the (finite) elements in a data type!

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

Base cases: We prove that $P(e_1), \dots, P(e_m)$;

Inductive steps: Assuming $P(s_1), \dots, P(s_{n_i})$ (our *inductive hypotheses* IH),
we prove $P(\mathsf{c}_1[s_1, \dots, s_{n_1}]), \dots, P(\mathsf{c}_\mathsf{k}[s_1, \dots, s_{n_k}])$;

Closure: $\forall s \in S.\ P(s)$.
(We will usually omit this part.)

Inductive Sets and Structural Induction

Inductive definition of S:

$$\frac{}{e_1 \in S} \quad \cdots \quad \frac{}{e_m \in S} \qquad \frac{s_1, \ldots, s_{n_1} \in S}{\mathsf{c}_1[s_1, \ldots, s_{n_1}] \in S} \quad \cdots \quad \frac{s_1, \ldots, s_{n_k} \in S}{\mathsf{c}_\mathsf{k}[s_1, \ldots, s_{n_k}] \in S}$$

Inductive principle associated to S:

$$\frac{\text{base cases} \begin{cases} P(e_1) \\ \vdots \\ P(e_m) \end{cases} \quad \text{inductive steps} \begin{cases} \forall s_1, \ldots, s_{n_1} \in S.\ \overbrace{P(s_1), \cdots, P(s_{n_1})}^{\text{IH}} \Rightarrow P(\mathsf{c}_1[s_1, \ldots, s_{n_1}]) \\ \vdots \\ \forall s_1, \ldots, s_{n_k} \in S.\ P(s_1), \cdots, P(s_{n_k}) \Rightarrow P(\mathsf{c}_\mathsf{k}[s_{k_1}, \ldots, s_{n_k}]) \end{cases}}{\forall s \in S.\ P(s)}$$

Example: Structural Induction over Lists

We can now use recursion to define functions over an inductively defined set and then prove properties of these functions by structural induction.

Let us (recursively) define the append and length functions over lists:

$$\begin{aligned} [] ++ ys &= ys \\ (a : xs) ++ ys &= a : (xs ++ ys) \end{aligned} \qquad \begin{aligned} \text{len } [] &= 0 \\ \text{len } (a : xs) &= 1 + \text{len } xs \end{aligned}$$

Proposition: $\forall xs, ys \in \text{List } A.\ \text{len } (xs ++ ys) = \text{len } xs + \text{len } ys$.

Proof: By structural induction on $xs \in \text{List } A$.
$P(xs)$ is $\forall ys \in \text{List } A.\ \text{len } (xs ++ ys) = \text{len } xs + \text{len } ys$.

Base case: We prove $P[]$.
Inductive step: We show $\forall xs \in \text{List } A, a \in A. P(xs) \Rightarrow P(a : xs)$.
Closure: $\forall xs \in \text{List } A.\ P(xs)$.

Example: Structural Induction over Lists

Let us (recursively) define the append and reverse functions over lists:

$$[] ++ ys = ys \qquad \text{rev } [] = []$$
$$(a : xs) ++ ys = a : (xs ++ ys) \qquad \text{rev } (a : xs) = \text{rev } xs ++ [a]$$

Assume append is associative and that $ys ++ [] = ys$.

Proposition: $\forall xs, ys \in \text{List } A.\ \text{rev } (xs ++ ys) = \text{rev } ys ++ \text{rev } xs.$

Proof: By structural induction on $xs \in \text{List } A$.
$P(xs)$ is $\forall ys \in \text{List } A.\ \text{rev } (xs ++ ys) = \text{rev } ys ++ \text{rev } xs$.

Base case: We prove $P[]$.
Inductive step: We show $\forall xs \in \text{List } A, a \in A. P(xs) \Rightarrow P(a : xs)$.
Closure: $\forall xs \in \text{List } A.\ P(xs)$.

Example: Structural Induction over Trees

Let us (recursively) define functions counting the number of edges and of nodes of a tree:

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

Proposition: $\forall t \in \text{Tree } A.\ \text{nn}(t) = 1 + \text{ne}(t).$

Proof: By structural induction on $t \in \text{Tree } A$.
$P(t)$ is $\text{nn}(t) = 1 + \text{ne}(t)$.

Base case: We prove $P(x)$.
Inductive step: We show $\forall t_1, \ldots, t_k \in \text{Tree } A, x \in A. P(t_1), \ldots, P(t_k) \Rightarrow P(x, t_1, \ldots, t_k)$.
Closure: $\forall t \in \text{Tree } A.\ P(t)$.

Proofs by Induction: Overview of the Steps to Follow

1. State property P to prove by induction.
   Might be more general than the actual statement we need to prove!
2. **Determine and state the method to use in the proof!!!!**
   Example: Mathematical induction on the length of the list, course-of-values induction on the height of a tree, structural induction over a certain element, ...
3. Identify and state base case(s).
   Could be more than one! Not always trivial to determine.
4. Prove base case(s).
5. Identify and state IH!
   Will depend on the method to be used (see point 2).
6. Prove inductive step(s).
7. (State closure.)
8. Deduce your statement from P (if needed).

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 $|\Sigma|$.

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

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

Example: Some alphabets:

- on/off-switch: $\Sigma = \{\text{Push}\}$;
- complex vending machine: $\Sigma = \{5\ kr, 10\ kr, \text{tea}, \text{coffee}\}$;
- parity counter: $\Sigma = \{p_0, p_1\}$.

Strings or Words

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

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

Note: Words will represent the *behaviour* of an automaton.

Example: Some behaviours:

- on/off-switch: Push Push Push Push;
- comlex vending machine: 5 kr 5 kr coffee 10 kr coffee 5 kr tea;
- parity counter: $p_0 p_1$ or $p_0 p_0 p_0 p_1 p_1 p_0$.

Note: Some words do NOT represent *behaviour* though . . .

Example: complex vending machine: tea 5 kr coffee.

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. Base case: $\epsilon \in \Sigma^*$;
   Inductive step: if $a \in \Sigma$ and $x \in \Sigma^*$ then $ax \in \Sigma^*$.
   (We will usually work with this definition.)
2. Base case: $\epsilon \in \Sigma^*$;
   Inductive step: if $a \in \Sigma$ and $x \in \Sigma^*$ then $xa \in \Sigma^*$.

Note: Recall the definition of lists!

Notation: We will often write just a instead of $a\epsilon$.

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

Concatenation

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

$$\epsilon y = y$$
$$(ax')y = a(x'y)$$

Note: Recall $++$ on lists.

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

If $x = 010$ and $y = 11$ then $xy = 01011$ and $yx = 11010$.

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

Prefix and Suffix

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 $y = zx$.

Note: $\forall\, x.\ \epsilon$ is both a prefix and a suffix of x.

Note: $\forall\, x.\ x$ is both a prefix and a suffix of x.

Length and Reverse

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

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

Example: $|01010| = 5$.

Definition: The *reverse* function $\text{rev}(\_) : \Sigma^* \to \Sigma^*$ as:

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

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

Note: Recall len and rev on lists.

Power

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

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

Example: $(010)^3 = 010010010$.

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

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

Some Properties

The following properties can be proved by induction:

Lemma: *Concatenation is associative:* $\forall x, y, z.\ x(yz) = (xy)z$.
We shall simply write xyz.

Lemma: $\forall x, y.\ |xy| = |x| + |y|$.
See proof on slide 5.

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

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

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

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

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

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: $\emptyset^* = \{\epsilon\}$ 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: $\mathcal{L}^+ = \mathcal{L}^1 \cup \mathcal{L}^2 \cup \mathcal{L}^3 \cup \ldots$

Example: Let $\mathcal{L} = \{aa, b\}$, then
$\mathcal{L}^0 = \{\epsilon\}$, $\mathcal{L}^1 = \mathcal{L}$, $\mathcal{L}^2 = \mathcal{L}\mathcal{L} = \{aaaa, aab, baa, bb\}$, $\mathcal{L}^3 = \mathcal{L}^2\mathcal{L}$, ...
$\mathcal{L}^* = \{\epsilon, aa, b, aaaa, aab, baa, bb, \ldots\}$.

How to Prove the Equality of Languages?

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

A few possibilities:

- Languages are sets so we prove that $\mathcal{M} \subseteq \mathcal{N}$ and $\mathcal{N} \subseteq \mathcal{M}$;
- Transitivity of equality: $\mathcal{M} = \mathcal{L}_1 = \ldots = \mathcal{L}_m = \mathcal{N}$;
- 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.

Algebraic Laws for Languages

All laws presented in slide 14 lecture 2 are valid.

In addition, we have all these laws on concatenation:

Associativity:	$\mathcal{L}(\mathcal{M}\mathcal{N}) = (\mathcal{L}\mathcal{M})\mathcal{N}$
Concatenation is *not commutative:*	$\mathcal{L}\mathcal{M} \neq \mathcal{M}\mathcal{L}$
Distributivity:	$\mathcal{L}(\mathcal{M} \cup \mathcal{N}) = \mathcal{L}\mathcal{M} \cup \mathcal{L}\mathcal{N} \quad (\mathcal{M} \cup \mathcal{N})\mathcal{L} = \mathcal{M}\mathcal{L} \cup \mathcal{N}\mathcal{L}$
Identity:	$\mathcal{L}\{\epsilon\} = \{\epsilon\}\mathcal{L} = \mathcal{L}$
Annihilator:	$\mathcal{L}\emptyset = \emptyset\mathcal{L} = \emptyset$
Other Rules:	$\emptyset^* = \{\epsilon\}^* = \{\epsilon\}$
	$\mathcal{L}^+ = \mathcal{L}\mathcal{L}^* = \mathcal{L}^*\mathcal{L}$
	$(\mathcal{L}^*)^* = \mathcal{L}^*$

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.

Example: Consider 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 satisfy

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

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

Note: $f(a) \in \Delta^*$ if $a \in \Sigma$.

Overview of next Week

Mon 27	Tue 28	Wed 29	Thu 30	Fri 31
	Ex 10-12 EA Proofs, induction.		**10-12 6213 6215 Individual help.**	
Lec 13-15 HB3 DFA.			**Lec 13-15 HB3** NFA. Subset construction.	
Ex 15-17 EC Proofs, induction.		**15-17 EL41 Consultation**		

Assignment 1: Formal proofs.
Deadline: Sunday April 2nd 23:59.

Overview of Next Lecture

Sections 2–2.2:

- DFA: deterministic finite automata.