Finite automata and formal languages

Notes based on lectures for DIT323 (Finite automata and formal languages)
at Gothenburg University, Spring 2024

Sebastian Pålsson

Last updated: **January 26, 2024**

Contents

1 Introduction

A **finite automaton** is a computational model with a set of states, input symbols, transition rules, an initial state, and accepting states. It recognizes patterns and processes strings in a language by transitioning between states based on input.

Formal languages are abstract systems defined by rules to represent and analyze languages. They provide a precise framework for specifying syntax, semantics, and rules for generating valid strings within a language.

1.1 Regular Expressions

Regular expressions are concise patterns for searching and matching strings, widely used in text processing and pattern matching.

- Used in text editors
- Used to describe the lexical syntax of programming languages.
- Can only describe a limited class of "languages".

Example:

- A regular expression for strings of ones of even length: $(11)^*$
- A regular expression for some keywords: ***while*** | ***for*** | ***if*** | ***else***
- A regular expression for positive natural number literals (of a certain form): $[1\text{–}9][0\text{–}9]^*$

1.2 Finite automata

- Used to implement regular expression matching.
- Used to specify or model systems.
  - One kind of finite automaton is used in the specification of TCP.
- Equivalent to regular expressions.

Example:

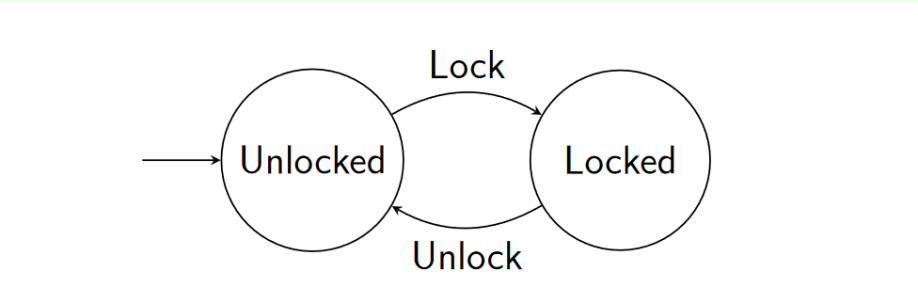


Figure 1: Model of a lock

- The states are a kind of memory.
- Finite number of states ⟹ finite memory.

1.3 Context-free grammars

Used to describe the syntax of programming languages.

- More general than regular expressions.
- Used by parser generators. (Often restricted.)

```
Expr ::== Number
    | Expr Op Expr
    | '(' Expr ')'
Op ::== '+' | '-' | '*' | '/'
```

1.4 Turing machines

A Turing machine is an abstract model of computation with a tape, read/write head, and rules. It serves as a foundational concept in the study of algorithms and computability.

- Unbounded memory: an infinite tape of cells.
- A read/write head that can move along the tape.
- A kind of finite state machine with rules for what the head should do.

It is equivalent to a number of other models of computation.

1.5 Repetition of some classical logic

1.5.1 Propositions

A **proposition** is a statement that is either true or false.

Example:

- The sky is blue.
- The sky is green.
- 1 + 1 = 2.
- 1 + 1 = 3.

It may not always be known what the truth value ($\top$ or $\bot$) of a proposition is.

1.5.2 Connectives

Logical connectives are operators that combine propositions to form new propositions.

$\mathbf{p} \wedge \mathbf{q}$	conjuction
$\mathbf{p} \vee \mathbf{q}$	disjunction
$\neg \mathbf{p}$	negation
$\mathbf{p} \Rightarrow \mathbf{q}$	implication
$\mathbf{p} \Leftrightarrow \mathbf{q}$	equivalence

Truth tables for these connectives:

$\mathbf{p}$	$\mathbf{q}$	$\mathbf{p} \wedge \mathbf{q}$	$\mathbf{p} \vee \mathbf{q}$	$\neg \mathbf{p}$	$\mathbf{p} \Rightarrow \mathbf{q}$	$\mathbf{p} \Leftrightarrow \mathbf{q}$
$\top$	$\top$	$\top$	$\top$	$\bot$	$\top$	$\top$
$\top$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\bot$
$\bot$	$\top$	$\bot$	$\top$	$\top$	$\top$	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\top$	$\top$	$\top$

Note that $\boldsymbol{p} \Rightarrow \boldsymbol{q}$ is true if $\boldsymbol{p}$ is false.

1.5.3 Validity

A proposition is *valid*, or a *tautology*, if it is satisfied for all assignments of truth values to its variables.

Example:
- $\boldsymbol{p} \Rightarrow \boldsymbol{p}$ is valid.
- $\boldsymbol{p} \vee \neg \boldsymbol{p}$ is valid

1.5.4 Equivalence

Two propositions are *equivalent* if they have the same truth value for all assignments of truth values to their variables. (they have the same truth table)

Example:
- $\boldsymbol{p} \Rightarrow \boldsymbol{q}$ and $\neg \boldsymbol{p} \vee \boldsymbol{q}$ are equivalent.
- $\boldsymbol{p} \wedge \boldsymbol{q}$ and $\boldsymbol{q} \wedge \boldsymbol{p}$ are equivalent.

1.5.5 Predicates

A predicate is, roughly speaking, a function to propositions.

Example:
- $P(n) =$ "n is a prime number"
- $Q(a, b) =$ "$(a + b)^2 = a^2 + 2ab + b^2$"

1.5.6 Quantifiers

Universal quantification and **existential quantification** are used to express statements about all or some elements in a set.

Example:
- $\forall \boldsymbol{n} \in \boldsymbol{N} : \boldsymbol{P}(\boldsymbol{n})$ means that $P(n)$ is true for all natural numbers n.
- $\exists \boldsymbol{n} \in \boldsymbol{N} : \boldsymbol{P}(\boldsymbol{n})$ means that $P(n)$ is true for some natural number n.

1.6 Repetition of some set theory

A set is roughly speaking a collection of elements.

1.6.1 Defining sets

- $\boldsymbol{A = \{1, 2, 3\}}$ means that A is the set containing the elements 1, 2, and 3.
- $\boldsymbol{B = \{x \in N \mid x > 0\}}$ means that B is the set of all natural numbers x such that $x > 0$.
- $\boldsymbol{C = \{x \in N \mid \exists y \in N : x = 2y\}}$ means that C is the set of all natural numbers x such that there exists a natural number y such that $x = 2y$. (the set of all even natural numbers)

1.6.2 Members, subsets, and equality

- $\boldsymbol{x \in A}$ means that x is an element of A.
- $\boldsymbol{A \subseteq B}$ means that A is a subset of B.
- $\boldsymbol{A = B}$ means that A and B are equal.

1.6.3 The empty set

- $\emptyset$ is the empty set.
- $\forall \boldsymbol{x} : \neg \boldsymbol{x} \in \emptyset$.

1.6.4 Set operations

Union, intersection and set difference

- $\boldsymbol{A \cup B}$ is the union of A and B. (the set of all elements that are in A or B)
- $\boldsymbol{A \cap B}$ is the intersection of A and B. (the set of all elements that are in A and B)
- $\boldsymbol{A \setminus B = A - B}$ is the set difference of A and B. (the set of all elements that are in A but not in B)

Complement

- $\boldsymbol{\overline{A}}$ is the complement of A. (the set of all elements that are not in A)

Cartesian product

- $\boldsymbol{A \times B}$ is the Cartesian product of A and B. (the set of all pairs (a, b) where $a \in A$ and $b \in B$)

> **Example**:
> - $\boldsymbol{N \times N}$ is the set of all pairs of natural numbers.
> - $\boldsymbol{N \times N \times N}$ is the set of all triples of natural numbers.

Power set

- $\boldsymbol{\wp(A) = \{A \mid A \subseteq S\}}$ is the power set of A. (the set of all subsets of A)

> **Example**:
> - $\wp(\{1, 2\}) = \{\emptyset, \{1\}, \{2\}, \{1, 2\}\}$

Set of all finite subsets

- $\boldsymbol{\mathbf{Fin}(A) = \{A \mid A \subseteq S, A \text{ is finite}\}}$ is the set of all finite subsets of A.

Example:
- $\mathbf{Fin}(\{\mathbf{1},\mathbf{2}\}) = \{\emptyset, \{\mathbf{1}\}, \{\mathbf{2}\}, \{\mathbf{1},\mathbf{2}\}\}$
- $\mathbf{Fin}(\boldsymbol{N}) = \wp(\boldsymbol{N})$

1.6.5 Relations

Relations define connections between elements of sets. A binary relation is a subset of the Cartesian product of two sets, often denoted as $R \subseteq A \times B$. Common types include reflexive, symmetric, and transitive relations, capturing different aspects of element connections.

- A binary relation R on A is a subset of $A^2 = A \times A : R \subseteq A^2$
- Notation: xRy same as $(x, y) \in R$
- Can be generalised from $A \times A$ to $A \times B \times C \times \cdots$

Some binary relational properties

For $R \subseteq A \times B$:

- Total (left-total): $\forall x \in A : \exists y \in B : xRy$
- Functional/deterministic: $\forall x \in A : \forall y, z \in B : xRy \wedge xRz \Rightarrow y = z$

For $R \subseteq A^2$:

- Reflexive: $\forall x \in A : xRx$
- Symmetric: $\forall x, y \in A : xRy \Rightarrow yRx$
- Transitive: $\forall x, y, z \in A : xRy \wedge yRz \Rightarrow xRz$
- Antisymmetric: $\forall x, y \in A : xRy \wedge yRx \Rightarrow x = y$

Partial orders

A *partial order* is a relation that is reflexive, antisymmetric, and transitive.

Equivalence relations

An *equivalence relation* is a relation that is reflexive, symmetric, and transitive.

1.6.6 Functions

Relation between two sets, denoted as $f : A \to B$, where A is the *domain* (set of inputs) and B is the *codomain* (set of possible outputs). Every element in the *domain* is associated with a unique element in the *codomain*. If (x, y) is in the function, it means that the input x is associated with the output y.

- Sometimes defined as the set of total and functional relations $f \subseteq A \times B$
- Notation $f(x) = y$ same as $(x, y) \in f$
- If the requirement of totality is dropped, we get the set of partial functions, $A \rightharpoonup B$
- The *image* is the set of all outputs of the function, $\{y \in B \mid x \in A, f(x) = y\}$

Identity, composition

- The *identity function* $\mathrm{id}_A : A \to A$ is defined as $\mathrm{id}_{A(x)} = x$
- For functions $f \in B \to C$ and $g \in A \to B$ the *composition* of $f \circ g \in A \to C$ is defined by $(f \circ g)(x) = f(g(x))$

Injectivity

An *injection* is a function $f : A \to B$ such that $\forall x, y \in A : f(x) = f(y) \Rightarrow x = y$

- Every input is mapped to an unique output.
- A is at most as large as B.
- Holds if f has a left inverse $g \in B \to A : g \circ f = id$

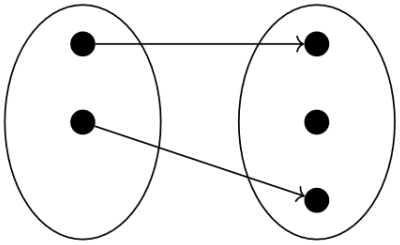


Figure 2: Injective function

Surjectivity

A *surjection* is a function $f : A \to B$ such that $\forall y \in B : \exists x \in A : f(x) = y$

- The function "targets" every element in the *codomain*
- A is at least as large as B.
- Holds if f has a right inverse $g \in B \to A : f \circ g = id$

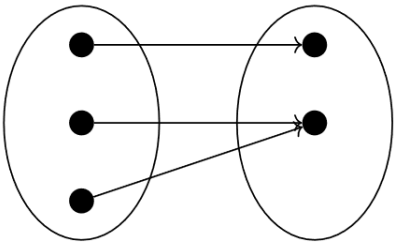


Figure 3: Surjective function

Bijectivity

A *bijection* is a function $f : A \to B$ such that $\forall y \in B : \exists! x \in A : f(x) = y$. *In simple terms, it is both injective and surjective.*

- A and B are of the same size.
- Holds *iff* f has left and right inverse $g \in B \to A$

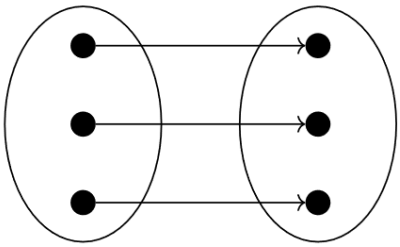


Figure 4: Bijective function

1.6.7 Partitions

A *partition* $P \subseteq \wp(A)$ of a set A is a set of non-empty subsets of A such that every element in A is in exactly one of the subsets.

- Every element is non-empty: $\forall X \in P : X \neq \emptyset$
- The elements cover A : $\bigcup_{B \in P} B = A$
- The elements are mutually disjoint: $\forall B, C \in P : B \neq C \Rightarrow B \cap C = \emptyset$

Example:
- $P = \{\{1, 2\}, \{3, 4\}\}$ is a partition of $A = \{1, 2, 3, 4\}$
- $P = \{\{1, 2\}, \{3, 4\}, \{5\}\}$ is not a partition of $A = \{1, 2, 3, 4\}$

1.6.8 Equivalence classes

Given a set A and an equivalence relation $R \subseteq A \times A$, the *equivalence class* of an element $a \in A$ is the set of all elements in A that are equivalent to a.

Definition 1.6.8.1: The equivalence classes of an equivalence relation R on A:

$$[x]_R = \{y \in A \mid xRy\} \tag{1}$$

1.6.9 Quotients

Given a set A and an equivalence relation $R \subseteq A \times A$, the quotient of A by R is the set of all equivalence classes of R.

Definition 1.6.9.1: The quotient of A by R is the set of all equivalence classes of R:

$$\frac{A}{R} = \left\{[x]_R \mid x \in A\right\} \tag{2}$$

Example: Can one define $\mathbb{Z} = \mathbb{N}^2$ with the intention that (m, n) stands for $m - n$?

No, $(0, 1)$ and (1,2) would both represent -1.

Instead we can use the quotient set:

$$\mathbb{Z} = \mathbb{N}^2 \setminus \underset{\mathbb{Z}}{\sim} \tag{3}$$

where

$$(m_1, n_2) \underset{\mathbb{Z}}{\sim} (m_2, n_2) \Leftrightarrow m_1 + n_1 = m_2 + n_2 \tag{4}$$

2 Proofs, induction & recursive functions

2.1 Basic proof methods

To prove	Method
$p \Rightarrow q$	Assume p and prove q
$p \Rightarrow q$	Assume ¬ q and prove $\neg p$
$\forall x \in A.P(x)$	Assume that we have an $x \in A$ and prove $P(x)$
$p \Leftrightarrow q$	Prove both $p \Rightarrow q$ and $q = p$
$\neg p$	Assume p and derive a contradiction
p	Prove $\neg\neg p$

2.2 Induction

For a natural number predicate P we can prove $\forall n \in \mathbb{N} : P(n)$ in the following way:

- Prove $P(0)$
- Prove $\forall n \in \mathbb{N} : P(n) \Rightarrow P(n+1)$

with the formula:

$$P(0) \wedge (\forall n \in \mathbb{N} : P(n) \Rightarrow P(n+1)) \Rightarrow \forall n \in \mathbb{N} : P(n) \quad [5]$$

2.3 Complete/Strong induction

We can also prove $\forall n \in \mathbb{N} : P(n)$ in the following way:

- Prove $P(0)$
- For every $n, i \in \mathbb{N}$, prove that if $P(i)$ holds for all $i \leq n$, $P(n+1)$ holds.

with the formula:

$$P(0) \land (\forall n \in \mathbb{N} : (\forall i \in \mathbb{N} : i \leq n \Rightarrow P(i)) \Rightarrow P(n+1)) \Rightarrow \forall n \in \mathbb{N} : P(n) \quad [6]$$

Example:

Lemma 2.3.1: Every natural number $n \geq 8$ can be written as a sum of multiples of 3 and 5.

Proof: *(by complete induction)*

Let $P(n) \coloneqq n \geq 8 \Rightarrow \exists a, b \in \mathbb{N} : n = 3a + 5b$

We prove that $P(n)$ holds for all $n \in \mathbb{N}$ by complete induction on n:

Basis: $(n = 8, 9, 10)$

- $P(8)$ holds since $8 = 3 * 1 + 5 * 1$.
- $P(9)$ holds since $9 = 3 * 3 + 5 * 0$.
- $P(10)$ holds since $9 = 3 * 0 + 5 * 2$.

Inductive step: $P(8) \land P(9) \land P(10) \land \cdots \land P(k) \Rightarrow P(k+1)$ where $k \geq i$

Induction hypothesis: Assume $P(i)$ is true for all $i \in \mathbb{N}$ where $10 \leq i \leq k$

To prove: $P(k+1) \coloneqq k + 1 \geq 8 \Rightarrow \exists a, b \in \mathbb{N} : k + 1 = 3a + 5b$

$$\begin{aligned} k + 1 = 3a + 5b &\Leftrightarrow k - 2 = 3a + 5b - 3 \\ &\Leftrightarrow k - 2 = 3(a - 1) + 5b \\ &\Leftrightarrow k - 2 = 3a' + 5b \end{aligned} \quad [7]$$

$k - 2 \geq 8$ and $k - 2 \leq k$, so by the *induction hypothesis*, $P(k-2)$ holds, thus $P(k+1)$ holds.

□

2.4 Proof by counterexample

To prove that a statement is false, we can find a counterexample.

In general, to prove:

$$\begin{aligned}&\neg(\forall \text{ natural number predicates } P : P(0) \wedge \\ &(\forall n \in \mathbb{N} : n \geq 1 \wedge P(n) \Rightarrow P(n+1)) \Rightarrow \\ &\forall n \in \mathbb{N} : n \geq 1 \Rightarrow P(n))\end{aligned} \tag{8}$$

we assume:

$$\begin{aligned}&\forall \text{ natural number predicates } P : P(0) \wedge \\ &(\forall n \in \mathbb{N} : n \geq 1 \wedge P(n) \Rightarrow P(n+1)) \Rightarrow \\ &\forall n \in \mathbb{N} : n \geq 1 \Rightarrow P(n)\end{aligned} \tag{9}$$

and derive a contradiction.

Example: The following statement does not hold for $P(n) := n \neq 1$ and $n = 1$

$$P(0) \wedge (\forall n \in \mathbb{N} : n \geq 1 \wedge P(n) \Rightarrow P(n+1)) \Rightarrow \forall n \in \mathbb{N} : n \geq 1 \Rightarrow P(n) \tag{10}$$

The hypotheses hold, but not the conclusion.

Proof:

Let $P(n) := n \neq 1$

Base case: $P(0) := 0 \neq 1$

To disproof:

$$\begin{aligned}&\forall n \in \mathbb{N} : n \geq 1 \wedge P(n) \Rightarrow P(n+1) \\ &\forall n \in \mathbb{N} : n \geq 1 \wedge n \neq 1 \Rightarrow n + 1 \neq 1\end{aligned} \tag{11}$$

Let $n = 1$

$$\begin{aligned}&n \geq 1 \wedge P(n) \Rightarrow P(n+1) \\ &n \geq 1 \wedge P(1) \Rightarrow P(2) \\ &1 \geq 1 \wedge 1 \neq 1 \Rightarrow 1 + 1 \neq 1\end{aligned} \tag{12}$$

$P(1) := 1 \neq 1$ is a contradiction, so we are done.

□

2.5 Inductively defined sets

An inductively defined set is a set that is defined in such a way that its elements are generated by applying a set of rules or operations starting from some initial elements. The process of generating elements continues indefinitely, and it relies on a principle of induction.

- **Base Elements:** Specify a set of initial elements that belong to the set. These are the starting points for the construction of the set.
- **Inductive Rules:** Define rules or operations that allow you to generate new elements of the set based on the existing elements.
- **Closure under Induction:** if an element belongs to the set, all the elements generated from it using the inductive rules also belong to the set.

Example: An example of an inductively defined set is the set of natural numbers:

$$\frac{}{\text{zero} \in \mathbb{N}} \qquad \frac{n \in \mathbb{N}}{\text{suc}(n) \in \mathbb{N}} \tag{13}$$

Example: Another example, booleans:

$$\frac{}{\text{true} \in \mathit{Bool}} \qquad \frac{}{\text{false} \in \mathit{Bool}} \tag{14}$$

Example: Another example, finite lists:

$$\frac{}{\text{nil} \in \mathit{List}(A)} \qquad \frac{x \in A \qquad xs \in \mathit{List}(A)}{\text{cons}(x, xs) \in \mathit{List}(A)} \tag{15}$$

Note that "nil" stands for the empty list and "cons" stands for the construction of a list by adding an element to the front of an existing list (in haskell).

Some alternative notations for lists:

- $[\,] \Leftrightarrow \text{nil}$
- $x : xs \Leftrightarrow \text{cons}(x, xs)$
- $[1, 2, 3] \Leftrightarrow \text{cons}(1, \text{cons}(2, \text{cons}(3, \text{nil})))$

2.6 Recursive functions

A recursive function is a function that is defined in terms of itself. It is defined by a base case and a recursive case.

Example:

$$\begin{aligned} \mathit{length} \in \mathit{List}(A) \quad &\to \mathbb{N} \\ \mathit{length}(\text{nil}) \quad &= \text{zero} \\ \mathit{length}(\text{cons}(x, xs)) \quad &= \text{suc}(\mathit{length}(xs)) \end{aligned} \tag{16}$$

Example:

$$\begin{aligned} &f \in List(A) \times List(A) \to List(A) \\ &f(\text{nil}, ys) &&= ys \\ &f(\text{cons}(x, xs), ys) &&= \text{cons}(x, f(xs, ys)) \end{aligned} \quad [17]$$

2.7 Mutual induction

In general, a proof by mutual induction of a statement A consists in proving a stronger statement than A, usually a statement of the form $A \wedge B$, where B can be seen as an *"auxiliary statement"*. The advantage is that now one can use a **stronger induction hypothesis** than in a proof of the mere statement A by simple induction.

Example: *The following is a nice example of a property about natural numbers that cannot be proved by simple induction (at least, not in a natural way), but the proof by mutual induction is very easy, because it gives a stronger induction hypothesis.*

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

$$f(n) = \begin{cases} 0 & \text{, if } n = 0 \\ g(n-1) & \text{, otherwise} \end{cases} \qquad g(n) = \begin{cases} 1 & \text{, if } n = 0 \\ f(n-1) & \text{, otherwise} \end{cases}$$

$$h(n) = \begin{cases} 0 & \text{, if } n = 0 \\ 1 - h(n-1) & \text{, otherwise} \end{cases} \quad [18]$$

Let $P(n) := \forall n \in \mathbb{N} : h(n) = f(n)$.

It does not seem possible to prove the proposition by simple induction, the induction hypothesis in this case is too weak. But the proof is quite easy if you prove the following stronger statement, by (mutual) induction on $n \in \mathbb{N}$:

Let $P'(n) := \forall n \in \mathbb{N} : h(n) = f(n) \wedge h(n) = 1 - g(n)$.

Proof: By induction on $n \in \mathbb{N}$.

Basis: $f(0) = 0 = h(0)$ and $h(0) = 0 = 1 - 1 = 1 - g(0)$

Inductive step: By induction hypothesis (i.h), $h(n) = f(n)$ and $h(n) = 1 - g(n)$. Hence,

$$\begin{aligned} &f(n+1) = g(n) = 1 - h(n) = 1 - f(n) = h(n+1) \\ &h(n+1) = 1 - h(n) = 1 - f(n) = 1 - g(n+1) \end{aligned} \quad [19]$$

□

Note that in the inductive step, to prove that $f(n+1) = h(n+1)$ we use the inductive hypothesis about the second statement, i.e. $h(n) = 1 - g(n)$. Conversely, to prove that $h(n+1) = 1 - g(n+1)$ we use the inductive hypothesis about the first statement, i.e. $f(n) = h(n)$. This is the essence of mutual induction.

3 Structural induction & automata theory

3.1 Structural induction

Structural induction is a proof technique that can be used to prove properties of recursively defined objects, such as inductively defined sets and recursive functions.

Example: For a given inductivley defined set we have a corresponding induction prionciple.

Consider the following inductively defined set of natural numbers:

$$\frac{}{\text{zero} \in \mathbb{N}} \qquad \frac{n \in \mathbb{N}}{\text{suc}(n) \in \mathbb{N}} \tag{20}$$

In order to prove $\forall n \in \mathbb{N} : P(n)$:

- Prove $P(\text{zero})$
- Prove $\forall n \in \mathbb{N} : P(n) \Rightarrow P(\text{suc}(n))$

Example: Another example, booleans:

$$\frac{}{\text{true} \in \mathit{Bool}} \qquad \frac{}{\text{false} \in \mathit{Bool}} \tag{21}$$

In order to prove $\forall b \in \mathit{Bool} : P(b)$:

- Prove $P(\text{true})$
- Prove $P(\text{false})$

Example: Another example, finite lists:

$$\frac{}{\text{nil} \in \mathit{List}(A)} \qquad \frac{x \in A \qquad xs \in \mathit{List}(A)}{\text{cons}(x, xs) \in \mathit{List}(A)} \tag{22}$$

In order to prove $\forall xs \in \mathit{List}(A) : P(xs)$:

- Prove $P(\text{nil})$
- Prove $\forall x \in A : \forall xs \in \mathit{List}(A) : P(xs) \Rightarrow P(\text{cons}(x, xs))$

The pattern of proving a proposition with structural induction.

Consider an inductivley defined set:

$$\ldots \quad \frac{x \in A \qquad d \in D(A)}{c(x, \ldots, d) \in D(A)} \quad \ldots \tag{23}$$

In order to prove $\forall d \in D(a) : P(d)$:

- $\vdots$
- $\forall x \in A, \ldots, d \in D(A)$, prove that ... and $P(d) \Rightarrow P(c(x, \ldots, d))$.
- $\vdots$

One inductive hypothesis for each recutsive argument.

3.2 Notes on induction/recursion

- *Inductively defined sets:* interference rules with constructors.
- *Recursion (primitive recursion):* recursive calls only for recursive arguments $f(c(x,d)) = \cdots f(d) \cdots$.
- *Structural induction:* inductive hypotheses for recursive arguments $P(d) \Rightarrow P(c(x,d))$.

3.3 Concepts from automata theory

3.3.1 Alphabets and strings

An alphabet is a finite, nonempty set of symbols.

- $\{a, b, c, \cdots, z\}$
- $\{0, 1, \cdots, 9\}$

A string (or word) over the alphabet is Σ is a member of *List*(Σ).

3.3.2 Some conventions

Following the course text book:

- Σ : An alphabet
- a, b, c : Elements of Σ
- u, v, w : Words (strings) over Σ

3.3.3 Simple notation

Σ^*	*List*(Σ)
ε	nil or [ ]
a w	cons(a, w)
a	cons(a, nil) or $[a]$
a, b, c	$[a, b, c]$
uv	append(u, v)
|w|	length(w)
σ^+	nonempty strings, $\{w \in \Sigma^* \mid w \neq \varepsilon\}$

3.3.4 Exponentiation

Σ^n : Strings of length n, $\{w \in \Sigma^* \mid |w| = n\}$

Example: $\{a, b\}^2 = \{aa, ab, ba, bb\}$

Alternative definition of $\Sigma^n \subseteq \Sigma^*$

- $\Sigma^0 = \{\varepsilon\}$
- $\Sigma^{\{n+1\}} = \{aw \mid a \in \Sigma, w \in \Sigma^n\}$

Repeated strings

w^n: w repeated n times

Example: $ab^2 = abbab$

Recursive definitions of functions on strings

- $w^0 = \varepsilon$
- $w^{\{n+1\}} = ww^n$

3.3.5 Languages

A language over an alphabet Σ is a subset $L \subseteq \Sigma^*$. Examples of such languages are typical programming languages such as C, Java, and Python and regular written languages. Another example is the odd natural numbers expressed in binary notation (without leading zeroes), which is a language over the alphabet $\{0, 1\}$.

Following the course text book: L, M, N : Languages over Σ.

3.3.6 Operations on languages

Some of these are mentioned earlier.

Operation	Rule	Example
Concatenation	$LM = \{uv \mid u \in L, v \in M\}$	$\{a, bc\}\{de, f\} = \{ade, af, bcde, bcf\}$
Exponentiation	$L^0 = \{\varepsilon\}$, $L^{\{n+1\}} = LL^n$	$\{a, bc\}^2 = \{a, bc\}\left(\{a, bc\}^1\right)$ $= \{a, bc\}(\{a, bc\}\{\varepsilon\})$ $= \{a, bc\}\{a, bc\}$ $= \{aa, abc, bca, bcbc\}$
The *Kleene star*	$L^* = \bigcup_{n \in \mathbb{N}} L^n$	$\{a, bc\}^* = \{a, bc\}^0 \cup \{a, bc\}^1 \cup \{a, bc\}^2 \cup \cdots$ $= \{\varepsilon, a, bc, aa, abc, bca, bcbc, \cdots\}$

3.4 Inductivley defined subsets

One can define subsets of (say) Σ^* inductively. For instance, for $L \subseteq \Sigma^*$ we can define $L^* \subseteq \Sigma^*$ inductively as follows:

$$\frac{}{\varepsilon \in L^*} \qquad \frac{u \in L \quad v \in L^*}{uv \in L^*} \tag{24}$$

Note that there are no constructors (but in some cases it might make sense to name the rules).

Example: $aba \in \{a, ab\}^*$

Proof:

$$\frac{\overline{ab \in \{a, ab\}} \qquad \frac{\overline{a \in \{a,ab\}} \qquad \overline{\varepsilon \in \{a,ab\}}}{a \in \{a,ab\}^*}}{aba \in \{a, ab\}^*} \tag{25}$$

□

some more shit on recursion with subsets is to be added here!

4 Deterministic finite automata

A *DFA* specifies a language. In Figure 5, the language is : $\{11\}^* = \{\varepsilon, 11, 1111, \cdots\}$. One of many use cases of *DFAs* is the implementation of regular expression matching.

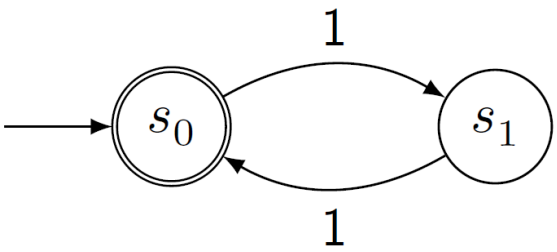


Figure 5: Transition diagram

A *DFA* is given by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

- Q : The finite set of states.
- Σ : The alphabet.
- $\delta \in Q \times \Sigma \to Q$: The transition function.
- $q_0 \in Q$: The starting state.
- $F \subseteq Q$: The set of accepting states

The diagram in Figure 5 corresponds to the 5-tuple

$$Even = (\{s_0, s_1\}, \{1\}, \delta, s_0, \{s_0\}) \tag{26}$$

where δ is defined in the following way:

$$\begin{aligned} &\delta \in \{s_0, s_1\} \times \{1\} \to \{s_0, s_1\} \\ &\delta(s_0, 1) = s_1 \\ &\delta(s_1, 1) = s_0 \end{aligned} \tag{27}$$

4.1 Semantics

Definition 4.1.1: The language $L(A)$ of a DFA $A = (Q, \Sigma, \delta, q_0, F)$ is defined in the following way:

A transition function for strings which is defined by recursion:

$$\begin{aligned} &\hat{\delta} \in Q \times \Sigma^* \to Q \\ &\hat{\delta}(q, \varepsilon) &&= q \\ &\hat{\delta}(q, aw) &&= \hat{\delta}(\delta(q, a), w) \end{aligned} \tag{28}$$

The language of A is defined as the set of all strings $w \in \Sigma^*$ such that $\hat{\delta}(q_0, w) \in F$:

$$L(A) = \left\{ w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F \right\} \tag{29}$$

Example: The language of the DFA in Figure 5 is specified as follows:

$$\begin{aligned}\hat{\delta}(s_0, 11) &= \hat{\delta}(\delta(s_0, 1), 1) \\ &= \hat{\delta}(s_1, 1) \\ &= \hat{\delta}(\delta(s_1, 1), \varepsilon) \\ &= \hat{\delta}(s_0, \varepsilon) \\ &= s_0\end{aligned} \quad [30]$$

4.2 Transition diagrams

A transition diagram is a graphical representation of a DFA. The states are represented by circles, the transitions by arrows, and the accepting states by double circles.

- One node per state.
- An arrow "from nowhere" to the starting state.
- Double circles for accepting states.
- For every transition $\delta(s_1, a) = s_2$, an arrow marked with a from s_1 to s_2. (multiple arrows can be combined)

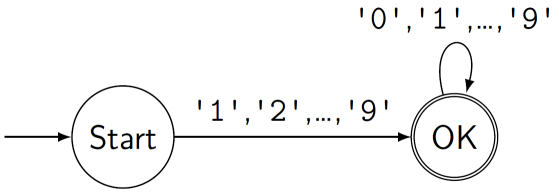


Figure 6: TD with missing transitions

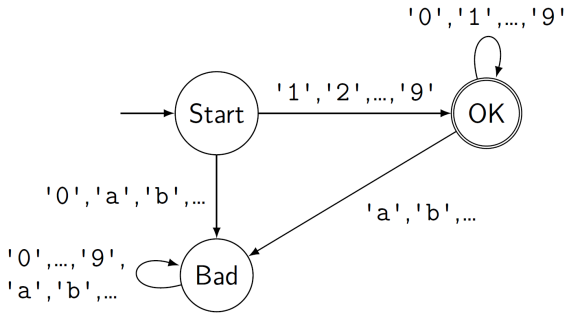


Figure 7: Every missing tranisition goes to a new state (that is not accepting)

Note that diagrams with missing tranisitions do not define the alphabet unambiguously. Consider the diagram in Figure 6, the alphabet must be a (finite) superset of $\{'0', '1', \cdots, '9'\}$*, but which one?*

4.3 Transition tables

A transition table is a tabular representation of a DFA. The states are represented by rows, the transitions by columns, and the accepting states by double entries.

		0	1
$\rightarrow$	$*s_0$	s_2	s_1
	s_1	s_2	s_0
	s_2	s_2	s_2

Figure 8: Transition table

- *States:* left column.
- *Alphabet:* top row.
- *Starting state:* marked with $\rightarrow$.
- *Accepting states:* marked with $*$
- *Transition function:* table.

4.4 Constructions

Given a DFA $A = (Q, \Sigma, \delta, q_0, F)$ we can construct a DFA $\overline{A}$ that satisfies the following property:

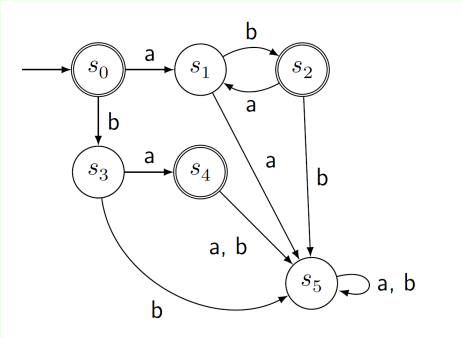
$$L\left(\overline{A}\right) = \overline{L(A)} := \Sigma^* \setminus L(A) \quad [31]$$

Construction:

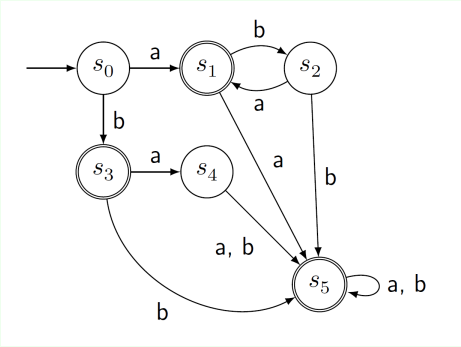
$$\overline{A} = (Q, \Sigma, \delta, q_0, Q \setminus F) \quad [32]$$

We accept if the original automaton doesn't. (what???)

Example: Let A be



Then $\overline{A}$ is



4.5 Product construction

Given two DFAs $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ and $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$ with the same alphabet, we can construct a DFA $A_1 \otimes A_2$ that satisfies the following property:

$$L(A_1 \otimes A_2) = L(A_1) \cap L(A_2) \quad [33]$$

Construction:

$$A_1 \otimes A_2 = (Q_1 \times Q_2, \Sigma, \delta, (q_{01}, q_{02}), F_1 \times F_2) \quad [34]$$

where

$$\delta((s_1, s_2), a) = (\delta_1(s_1, a), \delta_2(s_2, a)) \quad [35]$$

We basically run the two automatons in parallel and accept if both accept.

4.6 Sum Construction

Given two DFAs $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ and $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$ with the same alphabet, we can construct a DFA $A_1 \oplus A_2$ that satisfies the following property:

$$L(A_1 \oplus A_2) = L(A_1) \cup L(A_2) \quad [36]$$

Construction:

$$A_1 \oplus A_2 = (Q_1 \cup Q_2, \Sigma, \delta, (q_{01}, q_{02}), F_1 \cup F_2) \quad [37]$$

where

$$\delta((s_1, s_2), a) = \begin{cases} (\delta_1(s_1, a), s_2) & \text{, if } q \in Q_1 \\ (s_1, \delta_2(s_2, a)) & \text{, if } q \in Q_2 \end{cases} \quad [38]$$

4.7 Accessible states

Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA. The set $Acc(q) \subseteq Q$ of states that are accessible from $q \in Q$ can be defined in the following way:

$$Acc(q) = \left\{ \hat{\delta}(q, w) \mid w \in \Sigma^* \right\} \quad [39]$$

To construct a possible smaller DFA, which satisfies $L(A) = L(A')$:

$$\begin{aligned} A' &= (Acc(q_0), \Sigma, \delta', q_0, F \cap Acc(q_0)) \\ \delta'(q, a) &= \delta(q, a) \end{aligned} \quad [40]$$

Example: The following two DFAs define the same language:

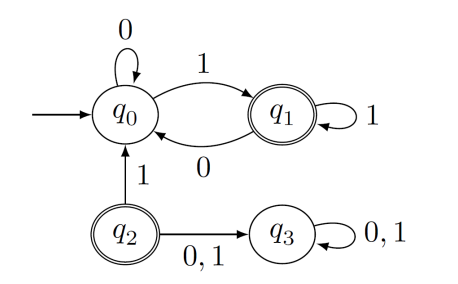


Figure 11: A

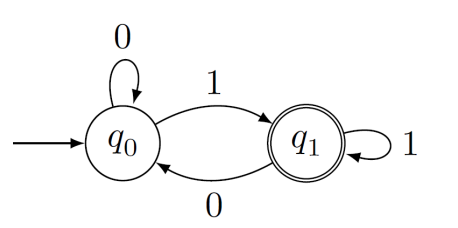


Figure 12: A'

Notice that A has unreachable states from the starting state q_0. We can ignore these states and construct A'.

4.8 Regular languages

A language $M \subseteq \Sigma^*$ is *regular* if there exists a DFA A with alphabet Σ such that $L(A) = M$.

- Note that if M and N are regular, then so are $M \cap N$, $M \cup N$, and $\overline{M}$.
- We will see later that if M and N are regular, then so are MN.

5 Non-deterministic finite automata & the subset construction