Models of Computation

CS 365

Eric Blais

LATEXed by Sibelius Peng

Preface

Disclaimer Much of the information on this set of notes is transcribed directly/indirectly from the lectures of CS 365 during Winter 2021 as well as other related resources. I do not make any warranties about the completeness, reliability and accuracy of this set of notes. Use at your own risk.

The original version of notes is available at https://cs.uwaterloo.ca/~eblais/cs365/. It was made with Book Theme: https://github.com/alex-shpak/hugo-book. This set of notes is more like a latex type-up of the original web version.

For any questions, send me an email via https://notes.sibeliusp.com/contact.

You can find my notes for other courses on https://notes.sibeliusp.com/.

Sibelius Peng

Contents

Preface 1

1 Languages 3
- 1.1 Alphabets, Strings, and Languages . . . 3
- 1.2 String Operations . . . 4
- 1.3 Language Operations . . . 5
- 1.4 Countability and Uncountability . . . 6

2 Deterministic Finite Automata 7
- 2.1 Nondeterministic Finite Automata . . . 8
- 2.2 Rabin-Scott Theorem . . . 9
- 2.3 Regular Expressions . . . 9
- 2.4 Kleene's Theorem . . . 9
- 2.5 The Pumping Lemma . . . 10
- 2.6 Non-Regular Languages . . . 10
- 2.7 Properties of Regular Languages . . . 11

3 Context-Free Languages 12

1

Languages

1.1 Alphabets, Strings, and Languages

All areas of mathematics are concerned with the study of the properties of specific types of mathematical objects: integers for number theory, graphs for graph theory, vector spaces for linear algebra, etc.

So what is the mathematical object that we study in theoretical computer science? It might be tempting to answer “computers”, or “algorithms”, or “computational problems”, but none of these answers is really accurate, simply because these concepts are not precise enough to enable rigorous mathematical analysis.

Instead, the main objects of study in theoretical computer science are much simpler: alphabets, strings, and languages. We begin by introducing these three (deceptively simple-looking) concepts.

alphabet

An **alphabet** is a nonempty finite set.

The elements of an alphabet are called **symbol**s.

string

A **string** over alphabet Σ is a finite sequence of symbols from Σ.

The **length** of the string x, denoted $|x|$, is the number of symbols in x.

The **empty string** is the unique string of length 0; we denote it by ε.

An **encoding** is a mapping from the objects to the set of strings such that no two objects are mapped to the same string.

For any non-empty string x of length $|x| = n$, we write $x = (x_1, \ldots, x_n)$ so that x_I is the ith symbol in x for any $i = 1, \ldots, n$.

language

A **language** over alphabet Σ is a set of strings over Σ.

A language is **finite** if it contains a finite number of strings. The **cardinality of a finite language** L is the number of strings that it contains and is denoted $|L|$.

The **empty language** $\varnothing$ is the unique language that contains no strings.

The language that contains every string over Σ is denoted Σ^*.

The language that contains the strings of length exactly n over Σ is denoted Σ^n.

1.2 String Operations

concatenation

The **concatenation** of the strings $x, y \in \Sigma^*$ of lengths $|x| = k$ and $|y| = \ell$ is the string

$$xy = (x_1, x_2, \ldots, x_k, y_1, y_2, \ldots, y_\ell).$$

square

The **square** of $x \in \Sigma^*$ is the string $x^2 = xx$.

More generally, for any $n \geq 1$ the nth power of x is string x^n obtained by concatenating n copies of x.

The 0th power of x is $x^0 = \varepsilon$.

primitive string

A string $x \in \Sigma^*$ is **primitive** if $x \neq y^n$ for any $y \in \Sigma^*$ and $n \geq 2$.

For any strings $x, y \in \Sigma^*$:

prefix

x is a **prefix** of y if there exists $z \in \Sigma^*$ such that $y = xz$.

suffix

x is a **suffix** of y if there exists $z \in \Sigma^*$ such that $y = zx$.

substring

x is a **substring** of y if there exists $w, z \in \Sigma^*$ such that $y = wxz$.

subsequence

The string x is a **subsequence** of the string y if there exist indices $j_1 < j_2 < \cdots < j_{|x|}$ such that $x_i = y_{j_i}$ for each $i = 1, 2, \ldots, |x|$.

reversal

The **reversal** of a string x of length $|x| = n$ is the string

$$x^R = (x_n, x_{n-1}, \ldots, x_2, x_1).$$

The string x is a **palindrome** if $x = x^R$.

1.3 Language Operations

Given two languages A and B over Σ,

- (Union) $A \cup B = \{x \in \Sigma^* : x \in A \text{ or } x \in B\}$;
- (Intersection) $(A \cap B) = \{x \in \Sigma^* : x \in A \text{ and } x \in B\}$;
- (Set difference) $A \setminus B = \{x \in \Sigma^* : x \in A \text{ and } x \notin B\}$;
- (Symmetric difference) $A \triangle B = (A \setminus B) \cup (B \setminus A)$; and
- (Complement) $\overline{A} = \Sigma^* \setminus A$.

concatenation

The **concatenation** of two languages A and B over Σ is the language

$$AB = \{xy : x \in A \text{ and } y \in B\}$$

obtained by concatenating every string in A with every string in B.

square

The **square** of a language L over Σ is the language $L^2 = LL$ obtained by concatenating L with itself.

More generally, for any language $n \geq 1$, the nth power of L is the language

$$L^n = \{x^{(1)}x^{(2)} \cdots x^{(n)} : x^{(1)}, x^{(2)}, \ldots, x^{(n)} \in L\}$$

obtained by concatenating n copies of L.

The 0th power of L is defined to be $L^0 = \{\varepsilon\}$.

start operation

For any language L, the **start operation** on L yields the language

$$L^* = \bigcup_{n \geq 0} L^n.$$

The star operation is also called the Kleene star operation, after Stephen Kleene who introduced it.

1.4 Countability and Uncountability

A really important notion in the study of the theory of computation is the uncountability of some infinite sets, along with the related argument technique known as the diagonalization method.

$|S| \le |T|$

The cardinalities of two sets S and T, denoted $|S|$ and $|T|$, satisfy the inequality $|S| \le |T|$ if and only if there is a one-to-one mapping from the elements of S to those of T.

same cardinality

Two sets S and T have the **same cardinality**, denoted $|S| = |T|$ if and only if $|S| \le |T|$ and $|T| \le |S|$.

The set if natural numbers is $\mathbb{N} = \{1, 2, 3, \dots\}$. Note that in CS 360, 0 is included.

finite nonempty set

A nonempty set S is **finite** if $|S| = |\{1, 2, \dots, n\}|$ for some natural number $n \in \mathbb{N}$. When this is the case, we write $|S| = n$ and say that S has cardinality $|S| = n$.

The empty set has cardinality $|\varnothing| = 0$ and is also finite.

The cardinality of set $\mathbb{N}$ of natural numbers is denoted $|\mathbb{N}| = \aleph_0$.

countable set

The set S is **countable** if and only if $|S| \le |\mathbb{N}| = \aleph_0$.

Proposition 1.1

For any alphabet Σ, the set Σ^* of strings over Σ is countable.

A set is **uncountable** if it is not countable.

power set

For any set S, the **power set** $\mathcal{P}(S)$ is the set of all subsets of S.

Theorem 1.2: Cantor's Theorem

For any infinite countable set S, the power set $\mathcal{P}(S)$ is uncountable.

Corollary 1.3

For any alphabet Σ, the set of languages over Σ is uncountable.

2

Deterministic Finite Automata

deterministic finite automation

A **deterministic finite automation** (or DFA) is an abstract machine described by

$$M = (Q, \Sigma, \delta, q_0, F)$$

where

- Q is a finite set of states,
- Σ is the input alphabet,
- $\delta : Q \times \Sigma \to Q$ is the transition function,
- $q_0 \in Q$ is the start state, and
- $F \subseteq Q$ is the set of accepting states.

The **size of the DFA** $M = (Q, \Sigma, \delta, s, F)$ is the number of states in Q.

DFA acceptance

The DFA $M = (Q, \Sigma, \delta, s, F)$ accepts the string $w \in \Sigma^n$ if and only if there is a sequence of states $r_0, r_1, \ldots, r_n \in Q$ where

- $r_0 = q_0$,
- $r_i = \delta(r_{i-1}, w_i)$ for each $i = 1, 2, \ldots, n$, and
- $r_n \in F$.

A string is rejected by a DFA if it is not accepted by that DFA.

language recognized by DFA

The **language recognized** by a DFA M is

$$L(M) = \{x \in \Sigma^* : M \text{ accepts } x\}$$

Note that in CS 360, this is written as $L(M)$.

regular language

The language A is **regular** if there is a DFA M such that $A = L(M)$.

Proposition 2.1

Every finite language is regular.

Proposition 2.2

There exist languages that are not regular.

2.1 Nondeterministic Finite Automata

nondeterministic finite automaton

A **nondeterministic finite automaton** (or NFA) is an abstract machine described by

$$M = (Q, \Sigma, \delta, q_0, F)$$

where

- Q is a finite set of states,
- Σ is the input alphabet,
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q)$ is the transition relation,
- $q_0 \in Q$ is the start state, and
- $F \subseteq Q$ is the set of accepting states.

An ϵ-**transition** in the NFA M is a transition of the form $\delta(q, \epsilon)$ for some $q \in Q$. The NFA M can choose to follow this transition from q at any time, without processing any symbol from its input.

ϵ-reachable

A state $r \in Q$ is **ϵ-reachable** from a state $q \in Q$ in an NFA M if there are states $s_0, s_1, \dots, s_\ell$ for some $\ell \geq 0$ that satisfy $s_0 = q$, $s_\ell = r$, and $s_i \in \delta(s_{i-1}, \epsilon)$ for each $i = 1, 2, \dots, \ell$. The state $r \in Q$ is ϵ-reachable from a set $S \subseteq Q$ of states if it is reachable from some state $q \in S$.

language recognized by NFA

The **language recognized** by the NFA N is

$$L(N) = \{x \in \Sigma^* : N \text{ accepts } x\}.$$

Proposition 2.3

Every regular language can be recognized by an NFA.

2.2 Rabin-Scott Theorem

Theorem 2.4: Rabin-Scott Theorem

The set of languages that can be recognized by DFAs is exactly the same as the set of languages that can be recognized by NFAs.

Lemma 2.5

For every NFA N, there is a DFA MM that recognizes $L(N)$.

2.3 Regular Expressions

regular expression

A **regular expression** over the alphabet Σ is a nonempty string r over $\Sigma \cup \{|,^*, (,), \epsilon, \varnothing\}$ that satisfies one of the following conditions:</p>

- $r = \varnothing$,
- $r = \epsilon$,
- $r = \sigma$ for some $\sigma \in \Sigma$,
- $r = (s \mid t)$ for some regular expressions s, t,
- $r = (st)$ for some regular expressions s, t, or
- $r = (s^*)$ for some regular expression s.

length

The **length** of a regular expression r over Σ is the number of symbols from Σ in the string r.

2.4 Kleene's Theorem

Theorem 2.6: Kleene's Theorem

The language A is regular if and only if it can be described by a regular expression.

We can prove the theorem via two lemmas.

Lemma 2.7

For every regular expression r, the language $L(r)$ described by r can be recognized by an NFA.

Lemma 2.8

Every language L that can be recognized by a DFA can be described with a regular expression.

2.5 The Pumping Lemma

Lemma 2.9: pumping lemma for regular languages

For every regular language L, there is a number p such that for any string $s \in L$ of length at least p, we can write $s = xyz$ where</p>

- $|y| > 0$,
- $|xy| \leq p$, and
- For each $i \geq 0$, $xy^iz \in L$.

A language L that satisfies the condition of the lemma is said to satisfy the *pumping property*.

A value p for which L satisfies the pumping property is known as a *pumping length* of L.

The Pumping Lemma is useful for showing that some languages are non-regular because it can be restated equivalently in the following contrapositive form:

Lemma 2.10: Contrapositive Form of the Pumping Lemma

If for every positive integer p, there exists a string $s \in L$ of length at least p such that for every decomposition $s = xyz$ with $|y| > 0$ and $|xy| \leq p$, there exists a value $i \geq 0$ for which $xy^iz \notin L$, then L is not a regular language.

2.6 Non-Regular Languages

$L = \{0^n1^n :\geq 0\}$ is not regular.

$L_<\{0^m1^n : n > m \geq 0\}$ is not regular.

$L_>\{0^m1^n : m > n \geq 0\}$ is not regular.

$L = \{0^{2^n} : n \geq 0\}$ is not regular.

One important warning about the Pumping Lemma is that the converse of that lemma is not true: there are languages that are not regular but still satisfy the pumping property. The following language gives one such example.

Proposition 2.11

The language $L_{\text{pal}^+} = \{ww^Rx \in \Sigma_2^* : |w| \geq 1\}$ satisfies the pumping property but it is not regular.

Another way to state the limitation of the pumping lemma is that it gives a necessary property that must hold for a language AA to be regular (since all regular languages satisfy the pumping property), but that this property does not characterize the class of regular languages since it is not a sufficient property to guarantee that a language is regular.

So does there exist a (natural) property of languages that characterizes regular languages? The answer is yes. This result is known as the Myhill-Nerode Theorem and is one of the topics covered in CS 462.

2.7 Properties of Regular Languages

We have already seen that the class of regular languages is closed under the union operation: given two regular languages A and B, their union $A \cup B$ is also a regular language. It is also closed under the complement, concatenation, and star operations.

language expansion

The **expansion** of the language A over Σ is the language

$$A^{\uparrow} = \{x \in \Sigma^* : \exists y \in A, |y| = |x|, \mathrm{d}(x,y) \leq 1\}.$$

The class of regular languages is closed under language expansion.

Topological Separation

It is interesting to ask about the regularity of languages obtained by taking subsets of other languages. Since every finite language is regular, all languages have a subset that is a regular language. Can we say more? The most natural possible strengthening of this observation is to ask whether every infinite language contains a subset that is an infinite regular language. The answer to this question is no.

There exists a language L over Σ_2 such that every infinite language $L' \subseteq L$ is non-regular.

What about when L is regular? It's easy to show that in this case it always contains a regular language as a strict subset.

For every infinite regular language L, there is an infinite language $L' \subsetneq L$ that is regular.

If you end up with the same proof of the last proposition that I obtained, you might find it rather unsatisfying. If so, the next natural question to ask is whether every infinite regular language L can be partitioned into two languages $L' \subseteq L$ and $L \setminus L'$ that are both infinite regular languages. This question is a type of topological separation problem, and the answer is yes.

For every infinite regular language L, there is a regular language $L' \subseteq L$ such that L' and $L \setminus L'$ are both infinite regular languages.

3 Context-Free Languages

Looking back on the specific languages examined in that note, we can identify what appears to be the main limitation of finite automata causing this issue: to recognize languages like $\{0^n1^n : n \geq 0\}$, a finite automaton would needs access to some memory.

The pushdown automaton model the model of computation when we give memory to finite automata in the form of a stack. This model of computation can indeed recognize a richer class of languages. And it also possesses very interesting structural properties of its own.

Index

A

alphabet 3

C

cardinality of a finite language 4
concatenation 4, 5
countable set 6

D

deterministic finite automation 7
DFA acceptance 7

E

ϵ-reachable 8
ϵ-transition 8
empty language 4
empty string 3
encoding 3

F

finite language 4
finite nonempty set 6

L

language 4
language expansion 11
language recognized by DFA 7
language recognized by NFA 8
length 3, 9

N

nondeterministic finite automaton 8

P

palindrome 5
power set 6
prefix 4
primitive string 4

R

regular expression 9
regular language 8
reversal 5

S

same cardinality 6
size of the DFA 7
square 4, 5
start operation 5
string 3
subsequence 5
substring 4
suffix 4
symbol 3

U

uncountable 6