Caio Brighenti
COSC 304 - Computer Theory
Fall 2019
Midterm Study Guide

1 Sets

Sets are collections of values of one type with no internal structure. Sets have no repeated items and order does not matter.

1.1 Set Operations

- Union - $A \cup B = \{x : x \in A \text{ or } x \in B\}$
- Intersect - $A \cap B = \{x : x \in A \text{ and } x \in B\}$
- Concatenation - $AB = \{ab : a \in A \text{ and } b \in B\}$
- Cartesian Product - $A\text{x}B = \{(a, b) : a \in A \text{ and } b \in B\}$
- Complement - $A^c = \{x : x \notin A\}$
- Powerset - $P(A) = \{A' : A' \subset A\}$

1.2 Finite and Infinite Sets

Set A is **finite** if it can be put in 1-to-1 correspondence with an initial segment of $\mathbb{N}$, or is the empty set $\emptyset$. More intuitively, a set is finite if we can list the elements of A in finite time.

Set A is **countably infinite** if it can be put in 1-to-1 correspondence with *all* of $\mathbb{N}$. Intuitively, a procedure can be devised to list all elements, but this procedure will never finish.

Set A is **uncountably infinite** if it is not countable.

1.3 Relations and Functions

A **function** can be defined as a binary operation between two sets that given an input in set A produces an output in set B. There are three types of functions:

1. Total function: $A \rightarrow B$ means for every input A there is exactly one output B
2. Partial function: $A \rightharpoonup B$ means for every input A there is at most one output B
3. Multi function: $A — B$ means for every input A there are 0, 1, or many outputs in B

We also define the **identity function** such that for any type A, $f(A) = A$.

1.4 Cardinality of Sets

The cardinality of a set A, or $|A|$, is the total number of elements in A.

- Union - $|A \cup B| = |A| + |B| - |A \cap B|$
- Intersect - $|A \cap B| = |A| + |B| - |A \cup B|$
- Concatenation -

- Cartesian Product - $|A\text{x}B| = |A|\text{x}|B|$
- Powerset - $P(A) = 2^{|A|}$
- Total function - $|A \to B| = |B|^{|A|}$
- Partial function - $|A \rightharpoonup B| = |B + 1|^{|A|}$
- Multi function - $|A — B| = |P(B)|^{|A|}$

2 Alphabets and Languages

Alphabet A is a set of single characters. **Word** w in alphabet A is a string of characters from A. A^* denotes the set of all words in alphabet A. This operation is called *Kleene star*.

2.1 Regular Languages

A language is **regular** if it can be defined as follows.

- The empty language $\emptyset$ is regular
- For each $a \in \epsilon$ where ϵ is the alphabet in question, the singleton language $\{a\}$ is regular
- If A and B are regular, then $A \cup B$ is regular
- If A and B are regular, then $A \cdot B$ is regular
- If A is regular, then A^* is regular.

Regular languages can be expressed by **regular expressions**. For each rule above, we have a corresponding regular expression.

- $e \to \emptyset$
- $a \to \{a\}$
- $a \vee b \to \{a\} \cup \{a\} = \{a, b\}$
- $ab \to \{a\} \cdot \{b\} = \{ab\}$
- $a^* \to \{a\}^* = \{\emptyset, a, a^2, a^3 \cdots\}$

2.2 Pumping Lemma

All finite languages are regular. For infinite languages, we use the Pumping lemma to determine whether or not they are regular. Specifically, the Pumping lemma states that if L is regular and infinite, then there exists an $N > 0$ such that for all words $\{w : w \in L \wedge |w| \geq N\}$, w can be decomposed into $w = xyz$ such that

- $y \neq e$
- $|xy| \leq N$
- $wy^k z \in L$ for all $k \geq 0$

This is typically used to show something is *not* regular through proof by contradiction.[1]

[1]See worksheet problem 5 for an example.

3 Finite Automata

A **deterministic finite automata** consists of

- alphabet A
- set of states $S = S_0, S_1, S_2 \cdots S_n$
- transition function $f : S \text{ x } A \to S$

The transition function given a state S_i and input a changes the machine to state S_j. By adding

- start state S_0
- final states $F \subset S$

we create a f.a. that can be said to *accept a language*. The language accepted by a fa can be defined as $LM = \{w | S_0 \overset{w}{\to} \text{stops in final state}\}$

3.1 Deterministic versus Non-Deterministic

A deterministic fa is one where the number of arrows leaving each state is exactly equal to $|A|$. In a **non-deterministic** fa the number of arrows leaving a state is ≥ 0. Additionally, the same word may lead to different states from the same state. In terms of functions, we consider the transition function as $f : S \text{ x } A \to S$ for a dfa and $f : S \text{ x } A — S$ for a ndfa.

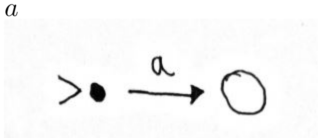
We describe a procedure for converting a ndfa into a dfa.[2]

1. List ES_0, the set including the initial state and all states reachable without consuming any characters
2. For $a \in A$, list the set of states reachable from ES_0 by consuming a
3. Repeat 2. until no new sets are produced (including Ø)
4. Draw the new dfa by treating each unique set as a state, where E_S0 is the initial state and any set containing a final state from the ndfa is a final state

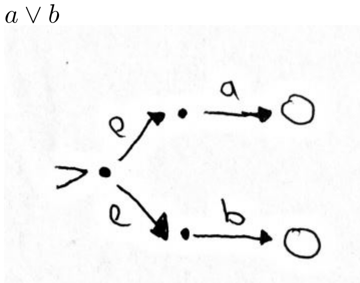
3.2 Regular Expressions to fa

The collection of languages accepted by fa is exactly equal to the regular languages. We can therefore define a simple fa for each of the regular expressions in section 2.1.

- a

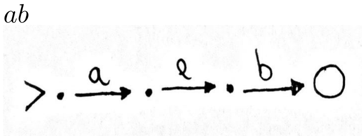


- $a \vee b$

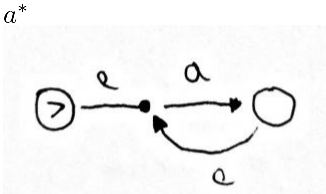


[2]See worksheet problem 7 for example.

- ab



- a^*



These can be used as building blocks to construct fa for more complex regular expressions. When two of these are chained together, a new state must be added where all final states can reach this new state using e.[3]

3.3 Simplifying dfa

We outline a process for simplifying a dfa. By simplifying we mean producing a new dfa with a possibly smaller number of states that accepts the same language.[4]

1. Partition the states into P_0 composed of two partitions, one with the final states F and the other with the non-final states F
2. List the behavior of each state $s \in S$ under each character $a \in A$
3. Partition P_0 into P_1 such that each state in each partition leads to the same partition when consuming $a \in A$ [5]
4. Repeat 3. until no new partitions are formed
5. Draw a dfa where each partition is a state, the partition including S_0 is the initial state, and any partition such that $P \subset F$ is a final state

3.4 Finding Language of f.a.

We define a recursive procedure for finding the language accepted by an arbitrary f.a.[6] For any two states i and j and intermediate states up to k, the words that take the f.a. from state i to j using intermediate states up to at most k is given by

$$R(i, j, k) = R(i, j, k-1) \cup R(i, k, k-1)R(k, k, k-1)^* R(k, j, k-1)$$

This is applied recursively to each term until k goes to 0, at which point we consider only edges. In practice it is often obvious that no paths exist prior to reaching $k = 0$, at which point the term can be substituted by $\emptyset$. It is also useful to note the following properties:

- $A \cdot \emptyset = \emptyset$, for any set A
- $\emptyset^* = \{e\}$
- $A \cdot \{e\} = A$, for any set A

[3] See worksheet problem 2 for example.
[4] See worksheet problem 8 for example.
[5] That is to say, all states in a partition must go to the same other partition given a character $a \in A$.
[6] See worksheet problem 4 for example.