Theory of Computation Notes

paraphrased by Tyler Wright

An important note, these notes are absolutely ***NOT*** *guaranteed to be correct, representative of the course, or rigorous. Any result of this is not the author's fault.*

Contents

1 The Basics of Computation

1.1 Decision Problems

A decision problem is a problem which has a `Yes` or `No` answer.

1.1.1 Decomposing Decision Problems

A decision problem can be decomposed into two sets, the `Yes` and `No` instances of the problem.

1.2 Alphabets

An alphabet is finite set whose members are called symbols (or equivalently letters or characters).

1.2.1 Strings

A string (or equivalently word) over an alphabet Σ is a finite sequence of symbols from Σ. The sequence may be empty, such sequences are denoted by ϵ. The amount of symbols in a string w is denoted by $|w|$.

1.2.2 The Set of Strings

The set of all strings over Σ is denoted by Σ^*.

1.2.3 Substrings and Concatenation

For two strings v, w, v is a substring of w if it appears consecutively in w.

We write vw to denotes v concatenated with w and for k in $\mathbb{Z}_{>0}$, we say v^k is the k-fold concatenation of v with itself (k copies of v).

2 Deterministic Finite State Automaton

A deterministic finite state automaton (DFA) is a 5-tuple $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ where:

$$\begin{aligned} Q &= \text{ any finite set, called the states,} \\ \Sigma &= \text{ any alphabet,} \\ \delta &\in \{Q \times \Sigma \to Q\} \text{ is the the transition function,} \\ q_0 &\in Q \text{ is the initial state,} \\ F &\subseteq Q \text{ is the set of accept states.} \end{aligned}$$

We say that M accepts a word w in Σ if there is a sequence of states $r_0, \ldots, r_n$ in Q satisfying:

- $r_0 = q_0$,
- $\delta(r_i, w_{i+1}) = r_{i+1}$,
- r_n is in F.

2.1 Product Automaton

For the two DFA:

$$M_1 = \langle Q_1, \Sigma, \delta_1, q_1, F_1 \rangle, M_2 = \langle Q_2, \Sigma, \delta_2, q_2, F_2 \rangle,$$

the product automaton M is:

$$M = M_1 \times M_2 = \langle Q, \Sigma, \delta, q_0, F \rangle,$$

where:

$$\begin{aligned} Q &= Q_1 \times Q_2 \\ \delta((p_1, p_2), a) &= (\delta_1(p_1, a), \delta_2(p_2, a)), \\ q_0 &= (q_1, q_2), \\ F &= F_1 \times F_2. \end{aligned}$$

3 Regular Languages

For a DFA M, the language set of M denoted by $L(M)$ is the maximal set of words in the alphabet of M such that for each w in $L(M)$, M accepts w. We say M recognises a language A if $L(M) = A$.

A language is regular if it is recognised by some DFA.

3.1 Operations

We can calculate the union and intersection of regular languages as expected and for two DFA M_1 and M_2 with languages A and B (resp.), we have that $A \cap B$ is recognised by $M_1 \times M_2$ the product automaton.

Additionally, we can concatenate two regular languages A and B:

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

and form the Kleene Star:

$$A^* = \{x_0 \cdots x_k : k \in \mathbb{Z}_{\geq 0} \text{ and for each } i \in \{0, 1, \ldots, k\}, x_i \in A\}.$$

We have that each of these operations are closed in the set of regular languages.

3.2 Regular Expressions

We have that R is a regular expression over an alphabet Σ if it has one of the following shapes:

$\emptyset$	
ϵ	
a	for some a in Σ
$R_1 \cup R_2$	for some regular expressions R_1 and R_2
$R_1 \circ R_2$	for some regular expressions R_1 and R_2
R^*	for some regular expression R

The language of regular expressions R_1 and R_2 can be formed as follows:

$$\begin{aligned} L(\emptyset) &= \emptyset \\ L(\epsilon) &= \{\epsilon\} \\ L(a) &= \{a\} \\ L(R_1 \cup R_2) &= L(R_1) \cup L(R_2) \\ L(R_1 \circ R_2) &= L(R_1) \circ L(R_2) \\ L(R_1^*) &= L(R_1)^* \end{aligned}$$

We have that a language L is regular if and only if $L = L(R)$ for some regular expression R.

3.3 Limitations of Regular Languages

We can see, for example, that $\{0^k 1^k : k \in \mathbb{Z}_{>0}\}$ over the alphabet $\{0, 1\}$ is not a regular language. By using the following Pumping Lemma, we can generate a contradiction.

3.3.1 The Pumping Lemma

Supposing A is regular, then there is some p in $\mathbb{Z}_{>0}$ such that for any word w longer than p, we can write $w = xyz$ such that:

- For each k in $\mathbb{Z}_{\geq 0}$, $xy^k z$ is in A,
- y is non-empty,
- xy is shorter than p.

4 Non-deterministic Finite State Automaton

A non-deterministic finite state automaton (NFA) is identical to a DFA except our transition function is from $Q \times \Sigma_\epsilon \to \mathcal{P}(Q)$ where Σ_ϵ is an alphabet Σ with the empty word added.

Transitioning on the empty word doesn't consume a letter of our input word and arbitrary choices are made by the automaton when choices present themselves. We have that a word is accepted in an NFA if and only if there is at least one computation where the word is accepted.

4.1 Epsilon Closure

For the NFA $M = \langle Q, \Sigma, \delta, q_0, F \rangle$, and $R \subseteq Q$, we define the epsilon closure of R to be:

$$E(R) := \left\{ q \in Q : \begin{array}{c} \text{where there is a series of transitions solely over} \\ \epsilon \text{ from some } r \text{ in } R \text{ to } q \end{array} \right\}$$

4.2 Simulation via a DFA

We can simulate an arbitrary NFA:

$$M = \langle Q, \Sigma, \delta, q_0, F \rangle$$

with a DFA:

$$M' = \langle Q', \Sigma_\epsilon, \delta', q_0', F' \rangle$$

where:

$$\begin{aligned} Q' &= \mathcal{P}(Q), \\ \delta'(q, a) &= \{q : \text{for some } r \in R, q \in E(\delta(r, a))\} \\ q_0' &= E(\{q_0\}), \\ F' &= \{q' \in Q' : \text{for some } q \in q', q \in F\}. \end{aligned}$$

Now that we have this, we know that languages are regular if and only if they are accepted by some NFA as all DFA are NFA and each NFA can be expressed by a DFA.

5 Generalised NFA

A generalised non-deterministic finite state automaton is a 5-tuple $M = \langle Q, \Sigma, \delta, p, p' \rangle$ where:

$$
\begin{aligned}
Q &= \text{ the set of states,} \\
\Sigma &= \text{ any alphabet,} \\
\delta &\in \left\{ (Q \setminus \{p'\}) \times (Q \setminus \{p\}) \to \mathcal{R} \right\} \text{ is the the transition function,} \\
p &\in Q \text{ is the initial state,} \\
p' &\in Q \text{ is the accept state,}
\end{aligned}
$$

where $\mathcal{R}$ is the set of all regular expressions. We say that that for a word $w = w_1 \cdots w_n$, M accepts w if for a corresponding series of states $q_0, \ldots, q_n$:

- $q_0 = p$,
- $q_n = p'$,
- for each i in $[n]$, w_i is in $L(\delta(q_{i-1}, q_i))$.

5.1 Conversion to a Regular Expression

We can distill a GNFA into a regular expression by iteratively removing states from it until there is only the start and accept state, joined by the sole transition which describes the GNFA as a regular expression.

Taking $M = \langle Q, \Sigma, \delta, p, p' \rangle$ to be a GNFA, we can choose q in $Q \setminus \{p, p'\}$ and form $M' = \langle Q \setminus \{q\}, \Sigma, \delta', p, p' \rangle$ where:

$$
\begin{aligned}
&\delta' : \left\{ (Q \setminus \{q, p'\}) \times (Q \setminus \{q, p\}) \to \mathcal{R} \right\} \\
&\delta'(q_1, q_2) = R_1 R_2^* R_3 \cup R_4,
\end{aligned}
$$

and R_1, R_2, R_3, R_4 are defined as:

$$
\begin{aligned}
R_1 &= \delta(q_1, q), \\
R_2 &= \delta(q, q), \\
R_3 &= \delta(q, q_2), \\
R_4 &= \delta(q_1, q_2).
\end{aligned}
$$

6 Context-free Grammars

We use context-free grammars to generate langauages. A context-free grammar is a 4-tuple $G = \langle V, \Sigma, R, S \rangle$ where:

$$\begin{aligned} V &= \text{ the set of variables (non-terminals)}, \\ \Sigma &= \text{ the set of terminals, disjoint from } V, \\ R &= \text{ the set of rules}, \\ S &\in V \text{ is the start variable.} \end{aligned}$$

We have that each rule is a pair of a variable A and a string w which it maps to.

By using the rules of the context-free grammar on the start variable, we can generate a language from it:

$$L(G) := \{w \in \Sigma^* : S \Rightarrow^* w\},$$

where $\Rightarrow^*$ denotes some amount of applications of the rules of G onto S.

6.1 Ambiguity

A derivation of a string w in a grammar G is a left-most derivation if, at every step in the derivation, the left-most remaining variable is evaluated.

A string w is generated ambiguously if it has more than one unique left-most derivation under G. G is ambiguous if it generates some string ambiguously.