Computational models, Computability and Complexity 67521

Based on lectures by Prof. Orna Kupferman, recitations by Ms. Maya Dotan and comments by Zip and Kent

Notes by Asaf Etgar
Spring 2021

Last edit: April 15, 2022
For any corrections, requests for additions or notes - please email me at
asafetgar@gmail.com

These notes have not been revised by the course staff, and some things may appear differently than in the lectures/ recitation.

Contents

Part I

Automata and Languages

Chapter 1

Regular languages

A finite automata is a finite state machine, designed to formalize the idea of "legal sequences of commands". To define an automata we would need some notion of commands, and a way to describe transitions between commands - in a way that captures (in)validity of a given sequence of commands.

1.1 Finite Automata

Definition 1.1 (Alphabet)**.** An **Alphabet** denoted by Σ is a finite set of letters $\Sigma = \{\sigma_1, \sigma_2 \dots \sigma_n\}$

Example 1.1. Some alphabet:

- $\Sigma = \{0, 1\}$
- $\Sigma = \{$On, Off, U,D,L,R$\}$

Definition 1.2 (Word, Empty Word)**.** A **Word** is a finite sequence of letters from a given alphabet. Denote the **Empty word** by ε. Denote the collection of all words over Σ by Σ^*, and note that $|\Sigma^*| = \aleph_0$ when Σ is finite.

Definition 1.3 (Language)**.** A **Language** is a subset of words $\mathcal{L} \subset \Sigma^*$ (we do not distinguish proper and improper inclusion).

Definition 1.4 (Finite Automaton)**.** A **Finite Automaton** consists of 5 items: $(Q, \Sigma, \delta, q_0, F)$, with:

- Q state set.
- Σ alphabet.
- $\delta : Q \times \Sigma \longrightarrow Q$ transition function.
- $q_0 \in Q$ initial state.
- $F \subset Q$ subset of accepting states.

Example 1.2. Define: $\Sigma = \{0, 1\}$, $Q = \{q_0, q_1\}$:

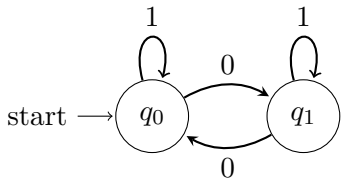


Figure 1.1: Simple Automaton

Note that here δ is implied, and that $F = \emptyset$. We may define $F = \{q_0\}$, and get the automaton:

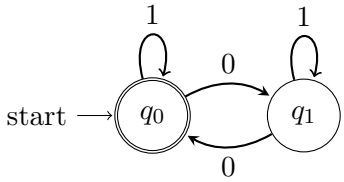


Figure 1.2: Simple Automaton with accepting state

Definition 1.5 (A Run on Automaton)**.** Let $\mathcal{A}$ be an automaton and $w = (w_i)_{i=1}^n \in \Sigma^*$ a word. A **Run** R on A is a sequence of states $R = (r_i)_{i=0}^n \in Q$ such that $r_0 = q_0$ and

$$\forall 0 \leq i \leq n-1 \qquad r_{i+1} = \delta(r_i, w_{i+1})$$

That is -R behaves according to δ (R respects δ).

Definition 1.6 (Accepting Run)**.** We say a run R is **accepting** if $r_n \in F$

Example 1.3. Recall the previous automaton with the word 1101, we get the run

$$q_0 \xrightarrow{1} q_0 \xrightarrow{1} q_0 \xrightarrow{0} q_1 \xrightarrow{1} q_1$$

Which is not accepting. Conversely - The run on 0101 is:

$$q_0 \xrightarrow{0} q_1 \xrightarrow{1} q_1 \xrightarrow{0} q_0 \xrightarrow{1} q_0$$

Which is accepting, because $q_0 \in F$.

Definition 1.7 (Accepting a word - deterministic)**.** We say an automaton $\mathcal{A}$ **Accepts** a word w if the (single) run R on w is accepting

Definition 1.8 (Language of Automaton)**.** The **Language of Automaton** $\mathcal{A}$ is:

$$\mathcal{L}(\mathcal{A}) = \{w \mid \mathcal{A} \text{ Accepts } w\}$$

Example 1.4. For the previous automaton $\mathcal{A}$, we have $\mathcal{L}(\mathcal{A}) = \{w \mid \text{the number of 0s in } w \text{ is even}\}$. Note that this is not a proof, but a claim. We will see later how to prove these claims (in various ways).

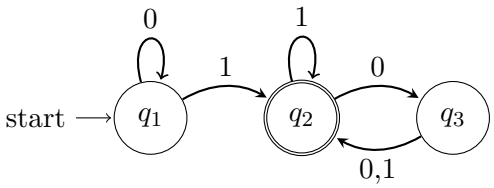


Figure 1.3: Example Automaton $\mathcal{A}$

Remark. Until now, we demanded that δ is defined for any $q \in Q, \sigma \in \Sigma$ (that is, A is **complete**). We could ignore this demand, and assume that for an incomplete automata - we could extend the incomplete δ to a complete one by introducing a **Rejecting Sink** $q_{rej} \in Q \setminus F$ and extend δ accordingly.

Example 1.5. Consider the following automaton What is $\mathcal{L}(\mathcal{A})$? Check a few words:

- 000 is not accepted
- 00010 is not accepted
- 01101 is accepted

The language is

$$L(A) = \{w \mid \text{There is at least one 1, and after the last 1 there is an even number of 0}\}$$

Example 1.6. Consider the following language:

$$\mathcal{L}_2 = \{w \in \{0,1\}^* \mid w \text{ contains } 001\}$$

We design the automaton:

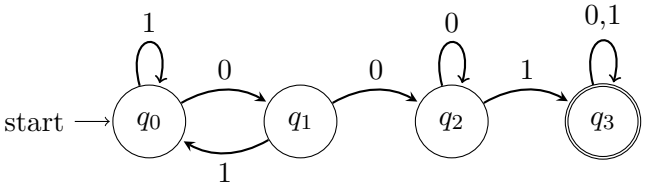


Figure 1.4: Automaton for $\mathcal{L}_2$

Example 1.7. We extend the last example, define:

$$\mathcal{L}_n = \{w \mid w \text{ contains } 0^n 1\}$$

A fitting automaton would be:

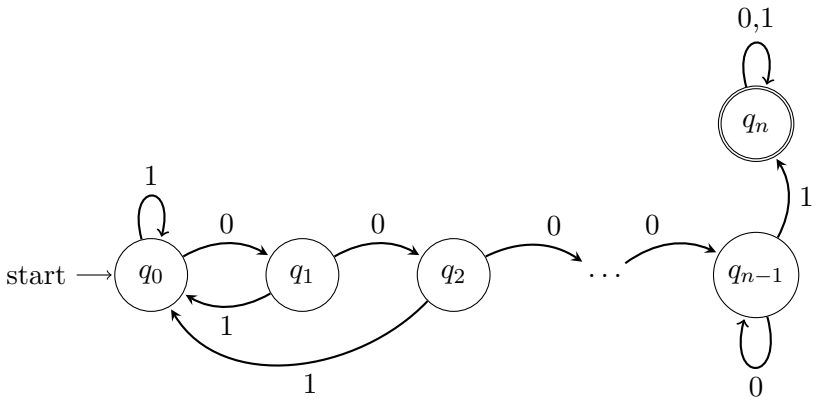


Figure 1.5: Automaton for $\mathcal{L}_n$

Definition 1.9 (δ^*)**.** : Given an automaton $\mathcal{A}$, we recursively define $\delta^* : Q \times \Sigma^* \to Q$:

$$\delta^*(Q, w) = \begin{cases} \delta(q, w) & |w| = 1, 0 \\ \delta\left(\delta^*(q, w'), \sigma\right) & w = w' \cdot \sigma \end{cases}$$

1.1.1 Deterministic Automata

Until now, we've designed automata for a given language. The natural question rises - is there an automaton for any language L? The answer is no, but it is not clear how to prove such a thing. At this point, let us define a language that is recognized by an automaton:

Definition 1.10 (Regular Language)**.** We say $L \subset \Sigma^*$ is **Regular** if there exists an automaton $\mathcal{A}$ over Σ such that $\mathcal{L} = \mathcal{L}(\mathcal{A})$. We denote the class of all regular languages **REG**.

1.1.2 Operations on Languages

Since any language $\mathcal{L}$ is a set, we may define some set operations on two languages $\mathcal{L}_1, \mathcal{L}_2$ - like union and intersection. But we can also define operations that use the concrete properties of languages.

Remark. We may always assume that $\mathcal{L}_1, \mathcal{L}_2$ are over the same $\Sigma = \Sigma_1 \cup \Sigma_2$.

Definition 1.11 (Concatenation)**.** We define the **Concatenation of $\mathcal{L}_1$ to $\mathcal{L}_2$** as

$$\mathcal{L}_1 \cdot \mathcal{L}_2 = \{w_1 \cdot w_2 \mid w_i \in \mathcal{L}_i \text{ for } i = 1, 2\}$$

Note that this operation is not commutative.

Definition 1.12 (Star)**.** For a language L define the **Star** operator as:

$$\mathcal{L}^* = \{w_1 \cdot w_2 \cdot \ldots \cdot w_k \mid k \geq 0, \forall i \leq k \quad w_i \in \mathcal{L}\}$$

Remark. Note that always $\varepsilon \in L^*$, and that Σ^* is a special case of this operation.

Definition 1.13 (Complement)**.** The **Complement** of L is:

$$\mathcal{L}^c = \overline{\mathcal{L}} = \Sigma^* \setminus \mathcal{L}$$

Example 1.8. Let $\mathcal{L}_1 = \{1, 333\}, \quad \mathcal{L}_2 = \{22, 4444\}$. Then:

- $\mathcal{L}_1 \cup \mathcal{L}_2 = \{1, 333, 22, 4444\}$
- $\mathcal{L}_1 \cdot \mathcal{L}_2 = \{122, 14444, 33322, 3334444\}$
- $\mathcal{L}_2 \cdot \mathcal{L}_1 = \{221, 22333, 44441, 4444333\}$
- $\mathcal{L}_1^*$ is an infinite set (countable).

1.1.3 Closure properties of regular languages

The natural question rises - if $\mathcal{L}_1, \mathcal{L}_2$ are regular, then which operations maintain the regularity property?

Theorem 1.1. *If $\mathcal{L}_1, \mathcal{L}_2$ are regular, then $\mathcal{L} := \mathcal{L}_1 \cup \mathcal{L}_2$ is also regular.*

Proof. Let $\mathcal{A}_i = (Q_1, \Sigma, \delta_1, s_1, F_1)$ $(i = 1, 2)$ such that $\mathcal{L}(\mathcal{A}_i) = \mathcal{L}_i$, we will construct $\mathcal{A} = (Q, \Sigma, \delta, s_0, F)$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}_1 \cup \mathcal{L}_2$. As always - we assume that δ_1, δ_2 are complete. Define $\mathcal{A}$ as the following (**product automaton**):

$$\begin{aligned} &Q = Q_1 \times Q_2, \quad s_0 = (s_1, s_2) \\ &\delta\left((q_1, q_2), \sigma\right)) = (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma)) \\ &F = \{(q_1, q_2) \mid q_1 \in F_1 \vee q_2 \in F_2\} = (F_1 \times Q_2) \cup (Q_1 \times F_2) \end{aligned}$$

Consider a run of $\mathcal{A}$ on a word w with $|w| = n$, denoted $R = q^0 \ldots q^n$, that is:

$$q^0, \ldots q^n = (q_1^0, q_2^0), (q_1^1, q_2^1), \ldots (q_1^n, q_2^n)$$

with transitions according to δ. By definition of δ, we have $q_1^i = \delta_1(q_1^{i-1}, w_i)$ and similarly for q_2^i. Thus, the run R defines two runs $R_1 = (q_1^i)_{i=0}^n, R_2 = (q_2^i)_{i=0}^n$ on $\mathcal{A}_1, \mathcal{A}_2$ for the word w (the converse claim also holds). The run R is accepting iff:

$$q^n \in F \iff (q_1^n, q_2^n) \in F \iff (q_1^n \in F_1 \vee q_2^n \in F_2)$$

that is - iff R_1 is accepting or R_2 is accepting. By definition, we have then $\mathcal{L}(\mathcal{A}) = \mathcal{L}_1 \cup \mathcal{L}_2$ □

Example 1.9. Consider $\Sigma = \{a\}$ and define $\mathcal{L}_1 = \{w \mid |w| \mod 2 = 0\}$ and $\mathcal{L}_2 = \{w \mid |w| \mod 2 = 1\}$, then:

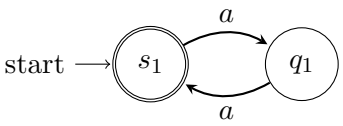


Figure 1.6: $\mathcal{A}_1$ Automaton for $\mathcal{L}_1$

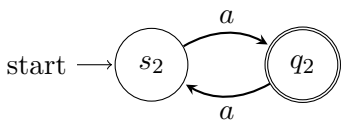


Figure 1.7: $\mathcal{A}_2$ Automaton for $\mathcal{L}_2$

Then the product automaton is:

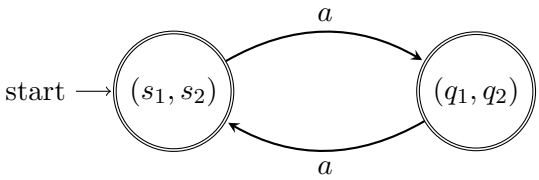


Figure 1.8: Automaton for $\mathcal{L}_1 \cup \mathcal{L}_2$: The product automaton of $\mathcal{A}_1$ and $\mathcal{A}_2$

Note that we only drew the **reachable** states in this automaton. Formally, the state (s_1, q_2) is a part of the automaton, but cannot be reached by any run on a given word (accepted or not). For simplicity - we omit unreachable states.

Theorem 1.2. *REG is closed under complementation, that is* $\mathcal{L} \in REG \implies \mathcal{L}^c \in REG$.

Theorem 1.3. *REG is closed under intersection, that is* $\mathcal{L}_1, \mathcal{L}_2 \in REG \implies \mathcal{L}_1 \cap \mathcal{L}_2 \in REG$.

Proof.

(Complementation:). *Let* $\mathcal{L}$ *be a regular language, and* $\mathcal{A} = (\Sigma, Q, F, \delta, q_0)$ *an automaton with* $\mathcal{L}(\mathcal{A}) = \mathcal{L}$. *Define the automaton* $\overline{\mathcal{A}}$ *the same as* $\mathcal{A}$, *but with* $F_{\overline{\mathcal{A}}} = Q \setminus F = \overline{F}$. *Thus:*

$$w \in \mathcal{L}(\overline{\mathcal{A}}) \iff \delta^*(q_0, w) \in \overline{F} \iff \delta^*(q_0, w) \notin F \iff w \notin \mathcal{L}(\mathcal{A}) \iff w \in \overline{\mathcal{L}(\mathcal{A})} = \overline{\mathcal{L}}$$

That is, by definition, $\mathcal{L}(\overline{\mathcal{A}}) = \overline{\mathcal{L}}$ *thus* $\overline{\mathcal{L}} \in REG$.

(Intersection:). *Let* $\mathcal{L}_1, \mathcal{L}_2 \in REG$. *By the previous theorem -* $\overline{\mathcal{L}_1}, \overline{\mathcal{L}_2} \in REG$. *We proved that REG is closed under union, so* $\overline{\mathcal{L}_1} \cup \overline{\mathcal{L}_2} \in REG$, *and once again by the previous theorem -* $\overline{\overline{\mathcal{L}_1} \cup \overline{\mathcal{L}_2}} = \mathcal{L}_1 \cap \mathcal{L}_2 \in REG$ *with the equality by de-Morgan's law. Thus REG is closed under intersection.*

□

We would like now to show that REG is closed under concatenation. This is not a trivial fact - in order to prove this, we will introduce a new concept.

1.2 Nondeterminism

Un until now, any DFA was, as the name suggests - deterministic. We will extend our discussion to the idea of non-deterministic automata. That is - for a tuple (q, σ) more than one transition is possible (more than one edge in the automaton graph) and in particular - edges $(q_1, \varepsilon) \to (q_2, \varepsilon)$ - "hops" between states, without reading anything.

1.2.1 Nondeterministic Automata

Definition 1.14 (Nondeterministic finite automaton)**.** a **Nondeterministic Finite Automaton (NFA)** $\mathcal{A}$ is a 5-tuple $(Q, \Sigma, \delta, Q_0, F)$ such that:

- Q, Σ, F as in DFA.
- $Q_0 \subset Q$ is a set of initial states
- $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$

Remark. Any DFA is a private case of NFA.

Remark. The "non-determinism" is captured by three facts:

- There may be more than one run for a word w on $\mathcal{A}$!
- There are ε transitions ("jumps")
- There are multiple initial states

Definition 1.15 (δ^*)**.** We extend δ to $\delta^* : 2^Q \times \Sigma^* \to 2^Q$ as

$$\delta^*(S, w) = \{\text{all states that are reachable by reading } w \text{ from } S\}$$

Formally we can compute δ^* inductively, starting from $w = \varepsilon$, then $|w| = 1$ and so on.

Example 1.10. Consider the following nondeterminisim automaton

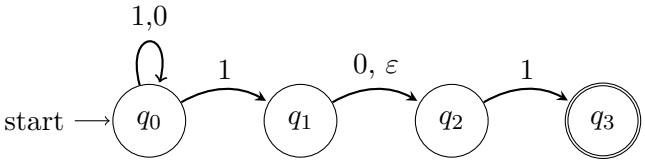


Figure 1.9: Nondeterministic Automaton

Definition 1.16 (Accepting run on NFA)**.** Given an NFA $\mathcal{A}$, we say that $w \in \mathcal{L}(\mathcal{A})$ if **there exists** a run $R = (r_i)_{i=1}^k$ of w on $\mathcal{A}$ such that $r_n \in F$.

Example 1.11. Consider the previous automaton, then $\mathcal{L}(\mathcal{A}) = \{\text{any word that contains 11 or 101}\}$. Once again - no formal proof here. AN important note - consider the word 11. The run (q_0, q_1) - it does not end in Q_0, but another run (q_0, q_1, q_2, q_3) is still valid - by using the ε transition, and this is an accepting run, so $11 \in \mathcal{L}(\mathcal{A})$.

1.2.2 Equivalence of NFAs and DFAs

Removing ε transitions

Claim 1.2.1. *Let $\mathcal{A}$ be an NFA, there exists an equivalent NFA $\mathcal{A}'$ with no ε transitions.*

Proof. Idea: For $q \in Q$ define

$$E_q = \{q' \in Q \mid q' \text{ is reachable from } q \text{ via } \varepsilon \text{ transitions}\}$$

Define $\mathcal{A}'$ the following way:

- $Q'_0 = \bigcup_{q \in Q_0} E_q$
- $\delta'(q, \sigma) = \bigcup_{s \in \delta(q,\sigma)} E_s$

□

Regulatity

As in DFA, we would like to understand for what languages $\mathcal{L}$ there exists an NFA $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}$. The surprising answer is that this class is exactly REG.

Theorem 1.4 (DFAs and NFAs Equivalence, Rabin and Scott 1959)**.** *For any NFA $\mathcal{A}$ there exists a DFA $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$.*

Remark. This means that for this computational model - nondeterminism gives no additional computational strength. This is not the case for all computational models, as we'll see later in the course.

Proof. Let $\mathcal{A}$ be an NFA. The idea would be to build a DFA $\mathcal{A}'$ (note its transition function ρ) to emulate any possible run of $\mathcal{A}$, and show $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$. We define $Q' = 2^Q$, $q'_0 = Q_0$ (initial states of $\mathcal{A}$), $F' = \{S \mid S \cap F \neq \emptyset\}$ and

$$\rho : Q' \times \Sigma \to Q' \qquad \rho(S, \sigma) = \delta^*(S, \sigma) = \bigcup_{t \in S} \delta(t, \sigma)$$

We prove $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$ by showing for any $w \in \Sigma^*$ we have $\rho^*(q_0, w) = \delta^*(Q_0, w)$, by induction on $|w|$:

(Base:)**.** *For $w = \varepsilon$, $\rho^*(q'_0, \varepsilon) = q'_0 = Q_0 = \delta^*(Q_0, \varepsilon)$*

(Step:)**.** *for $w = x \cdot \sigma$ we have:*

$$\rho^*(q'_0, w) = \rho\left(\rho^*(q'_0, x), \sigma\right) = \rho\left(\delta^*(Q_0, x), \sigma\right) = \\ \delta\left(\delta^*(Q_0, x), \sigma\right) = \delta^*(Q_0, x \cdot \sigma) = \delta^*(Q_0, w)$$

The claim follows by F''s definition:

$$w \in \mathcal{L}(A') \iff \rho^*(q_0, w) \cap F \neq \emptyset \iff \delta^*(Q_0, w) \cap F \neq \emptyset \iff w \in \mathcal{L}(\mathcal{A})$$

□

Remark. This is called the detrminization of $\mathcal{A}$, and it involves exponential extention: $\mathcal{A}$ with n states then $\mathcal{A}'$ with 2^n states.

Corollary 1.5. *REG is closed under concatenation.*

Proof. Let $\mathcal{L}_1, \mathcal{L}_2 \in$ REG and $\mathcal{A}_1, \mathcal{A}_2$ fitting DFAs. Let $\mathcal{A}$ be the following automaton: For any $q \in F_1$ define an ε transition to q_0^2 (initial state in $\mathcal{A}_2$), $F = F_2$, $Q_0 = \{q_0^1\}$. That is, assume $Q_1 \cap Q_2 = \emptyset$, and define:

$$\delta'(q, \sigma) = \begin{cases} \delta(q, \sigma) & q \in Q \\ \{S_0\} & \sigma = \varepsilon \\ \rho(q, \sigma) & q \in S \\ \emptyset & \text{otherwise} \end{cases}$$

Note that:

$$w \in \mathbb{L}_1 \cdot \mathcal{L}_2 \iff w = w_1 \cdot w_2 \iff \exists \text{two runs } r_1, r_2 \text{ s.t } r_1^n \in F_1, r_2^k \in F_2 \iff w \in \mathcal{L}(\mathcal{A})$$

Since $\mathcal{A}$ is an NFA with $\mathcal{L}(A) = \mathcal{L}_1 \cdot \mathcal{L}_2$, and the previous theorem - we have $\mathcal{L}_1 \cdot \mathcal{L}_2 \in$ REG □

Theorem 1.6 (Closure under star operation)**.** *REG is closed under star operation.*

Proof. Let $\mathcal{L} \in$ REG, and note that $\mathcal{L}^* = \bigcup_{k \in \mathbb{N}_0} \mathcal{L}^k$. Let $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}$. Define NFA $\mathcal{A}'$ for $\mathcal{L}^*$ the following way:

- $Q' = Q \cup \{q_{start}\}$
- $F = Q'_0 = \{q_{start}\}$

and:

$$\delta(q,\sigma) = \begin{cases} \{\delta(q,\sigma)\} & q \in Q, \sigma \in \Sigma \\ \emptyset & q = q_{start} \\ \emptyset & q \in Q \setminus F, \sigma = \varepsilon \\ \{q_{start}\} & q \in F, \sigma = \varepsilon \\ \{q_0\} & q = q_{start}, \sigma = \varepsilon \end{cases}$$

(The intuition here is that $\emptyset$ is a rejecting sink, and that any run that ends in F might "jump" to q_{start}). □

Exponential extention is best case

Theorem 1.7 (Lower bound for space complexity of determinization)**.** *Let $\Phi : NFA \to DFA$ a determinzation. Then there exists a family of regular languages $\mathcal{L}_k = \mathcal{L}(\mathcal{A}_k)$ such that:*

$$|\{\text{states of } \Phi(\mathcal{A}_k)\}| = \Omega\left(2^{|\{\text{states of } \mathcal{A}_k\}|}\right)$$

Proof. Construct a family of languages $\mathcal{L}_n$ for $n \in \mathbb{N}$ such that:

- $\mathcal{L}_n$ has NFA with $O(n)$ states.
- The smallest DFA for $\mathcal{L}_n$ needs at least 2^n states.

Over $\Sigma = \{0,1\}$, define $\mathcal{L}_n = \{w \in \Sigma^* \mid w_{|w|-(n)} = 1\}$ That is - all words with 1 in the n'th place from the end. An NFA for this is:

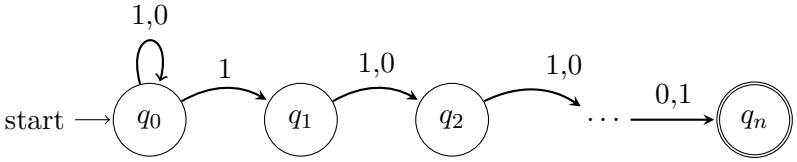


Figure 1.10: Automaton for $\mathcal{L}_n$

Asuume for sake of contradiction there exists a DFA $\mathcal{A}_n$ for $\mathcal{L}_n$ with less than 2^n states. There has to be two words w_0, w_1 such that $\mathcal{A}_n$ reaches the same state after reading both words

(by pigeonhole principle over $(0+1)^n$), that is $\delta^*(q_0, w_0) = \delta^*(q_0, w_1)$. Let i be such that $w_0[i] \neq w_1[i]$ and assome WLOG $w_0[i] = 0$. Consider that word $w' = w_0 \cdot 1^{i-1}$ (we want $w_0[i]$ to be in the n'th place from the end)[I]. Does $\mathcal{A}_n$ accepts w'? If yes - it's a contradiction for $\mathcal{A}_n$ as an automaton for $\mathcal{L}_n$. If not - then

$$\delta^*(\delta^*(q_0, w_1), 1^{i-1}) \overset{\text{by choice of } w_1, w_2}{=} \delta^*(\delta^*(q_0, w_0), 1^{i-1}) \notin F$$

Once again - a contradiction (since $w_1 1^{i-1} \in \mathcal{L}_n$). □

Remark. This is not the complete argument. One has to formalize this a little further (will leave empty to force myself to practice)

[I] Since $w_0 = (0+1)^{i-1}0(0+1)^{n-i}$, concatenating with a word of length $i-1$ puts $w_0[i]$ in the n'th place from the end.

1.3 Regular Expressions

1.3.1 Formal Definition of Regular Expressions

We would like to have a compact way of describing words and (regular) languages. That is - DFAs and NFAs are "checking devices", and not ways to *generate* languages. The generative model we use would be **Regular Expressions**.

Definition 1.17 (Regular Expressions)**.** We define a **Regular Expression over** Σ inductively:

(Base:)**.** $\varepsilon, \emptyset, \sigma$ *are regular expressions.*

(Step:)**.** *if* r_1, r_2 *are regexes, then:*

- $r_1 + r_2 = r_1 \cup r_2$ *("Or")*
- $r_1 \cdot r_2$ *("Concatenation", will usually omit this dot)*
- r_1^* *("Star")*

Are regexes.

Definition 1.18 (Language of Regex)**.** Let r be a regex. Then $\mathcal{L}(r)$ is defined by induction:

(Base:)**.** $\mathcal{L}(\varepsilon) = \{\varepsilon\}, \mathcal{L}(\emptyset) = \emptyset, \mathcal{L}(\sigma) = \{\sigma\}$

(Step:)**.** *if* r_1, r_2 *are regexes, then:*

- $\mathcal{L}(r_1 + r_2) = \mathcal{L}(r_1) \cup \mathcal{L}(r_2)$
- $\mathcal{L}(r_1 \cdot r_2) = \mathcal{L}(r_1) \cdot \mathcal{L}(r_2)$
- $\mathcal{L}(r_1^*) = \mathcal{L}(r_1)^*$

Remark. We denote $r^+ = r \cdot r^*$ (a non-empty concatenation)

Example 1.12. Assume we want words over Σ of even length. A regex would be $(\Sigma \cdot \Sigma)^* = ((0+1)\cdot(0+1))^*$

Example 1.13. Consider the condition "every words that has 0 in one of the last 3 characters". Then a fitting regex would be:

$$((0+1)^*0(0+1)(0+1)) + ((0+1)^*0(0+1)) + ((0+1)^*0)$$

Alternatively:

$$(0+1)^*0((0+1) + (0+1) + (0+1) + \varepsilon)$$

Alternatively:

$$(0+1)^*0(0+1+\varepsilon)(0+1+\varepsilon)$$

Example 1.14. Consider the complement of the previous regex, that is - words that don't have 0 in the last 3 places, a fitting regex would be

$$\varepsilon + 1 + 11 + (0+1)^*111$$

(which is simply considering private cases)

1.3.2 Equivalence with Finite Automata

It should come with no surprise that *regular* expressions have a deep connection to *regular* languages. What we wish to prove is that any regex in face describes a regular language (thus a finite automata), and vice versa.

Definition 1.19 (Generalized NFA)**.** A **Generalized NFA (GNFA)** is an NFA with regexes on the edges, instead of letters (that is, δ is a function $Q \times \text{REGEX} \to 2^Q$).

Remark. We assume that:

- Unique initial state that is a source.
- Unique accepting state that is a sink.
- $q_{start} \neq q_{end}$

Claim 1.3.1. *Generalized NFAs describe precisely the regular languages.*

Proof. No. □

Theorem 1.8 (Characterization of Regular Lnaguages)**.**

$$\mathcal{L} \in REG \iff \mathcal{L} = \mathcal{L}(r) \text{ for some regex } r$$

Proof.

(Easy side:)**.** *if $\mathcal{L} = \mathcal{L}(r)$ for some r regex - $\mathcal{L}$ is trivially regular (by induction on the regex's structure).*

(Hard side:)**.** *Let $\mathcal{L} \in REG$, we construct a regex r for $\mathcal{L}$. We would like to construct an algorithm that gets a DFA and outputs a regex. We will construct a GNFA equivalent to the DFA (Take a look at the example in the Recitation Notes):*

1. *Add s, t states (initial and accepting) with ε transitions from s to q_0 and from F to t.*
2. *Start removing states one by one, and "fixing" transitions by defining edges with appropriate regexes.*
3. *We finish with only two states - s, t with a single edge between them. The regex on that edge r satisfies $\mathcal{L}(r) = \mathcal{L}(\mathcal{A}) = \mathcal{L}$.*

□

1.4 Nonregular Languages

We claimed in the past that $\mathcal{L} = \{0^n 1^n \mid n \in \mathcal{N}_0\}$ is not regular. Now we will see how to formally prove this.

Claim 1.4.1. $\mathcal{L} \notin REG$

Proof. By contradiction. Assume $\mathcal{A}$ with p states is a DFA for $\mathcal{L}$. Consider the word $w = 0^p \cdot 1^p$. Since $w \in \mathcal{L}(\mathcal{A})$ there is an accepting run $(q_i)_{i \in [2p]}$. By pigeonhole principle, there are $0 \leq l < j \leq p$ s.t $q_j = q_l$ (since we read $p+1$ states until q_p). This means that $\mathcal{A}$ accepts the word $0^{p \pm (j-l)} q^p$ (draw this for intuition), contradicting the assumption. □

1.4.1 Pumping Lemma for regular languages

Theorem 1.9 (Pumping Lemma)**.** *Let $\mathcal{L} \in REG$, then there exists $p \geq 1$ (called the **pumping constant**) such that for $w \in \mathcal{L}$, if $|w| \geq p$, then there exists a partition of w to $w = x \cdot y \cdot z$ such that:*

1. *$|y| > 0$ (but we allow $x, z = \varepsilon$)*
2. *$|x \cdot y| \leq p$*
3. *For all $i \geq 0$ the word $x \cdot y^i z \in \mathcal{L}$*

Remark. This is equivalent to the following statement:
If for any $p \geq 1$ there exists a word $w \in \mathcal{L}$ with $|w| \geq p$ and for any partition of w to xyz, if $|y| > 0$ and $|x \cdot y| \geq p$ then there exists $i \geq 0$ such that $xy^i z \notin \mathcal{L}$ then $\mathcal{L}$ is not regular.
This is simply the counter-positive of the Pumping Lemma.

Remark. We will use this mainly to show that a language is not regular - that is, no pumping constant exists.

Proof. Take p to be the number of stats of $\mathcal{A}$ an automaton of $\mathcal{L}$, consider $w \in \mathcal{L}$ with $|w| > p$, then in a run $\mathcal{A}(w) = (q_i)_{i \in [n]}$ by pigeonhole principle, there exists[II] $0 \leq l < j \leq p$ with $q_l = q_j =: s$. Define

$$x = (w_i)_{0 < i \leq l}, y = (w_i)_{l < i \leq j}, z = (w_i)_{q < i \leq n}$$

The conditions hold: $|y| > 0$ by choice of $l < j$. $|xy| \leq p$ by choice of j, and $xy^i z \in \mathcal{L}$ by the run

$$\mathcal{A}(xy^i z) = (q_k)_{k \in [l]} \left[(q_k)_{l+1 \leq k \leq j}\right]^i (q_k)_{k \geq j+1}$$

As required. □

Example 1.15. Consider $\mathcal{L} = (0+1)^*0(0+1)$, we show that $p = e = \pi = 3$ is a pumping constant for $\mathcal{L}$. For any $w \in \mathcal{L}$ with $|w| \geq 3$, we choose $x = \varepsilon, y = w[1]$ and z be the suffix $(w_i)_{i \geq 2}$. One can verify that the conditions hold.

Example 1.16 (Applying the Pumping Lemma)**.** Let $\mathcal{L} = \{0^n 1^n \mid n \in \mathbb{N}\}$, we show $\mathcal{L}$ is not regular by the previous remark. For any $p \geq 1$ we show that the word $w = 0^p 1^p$ is "bad" for any partition. Consider $w = x \cdot y \cdot z$, and show that $x \cdot y^2 \cdot z \notin \mathcal{L}$:
Since $|xy| \leq p$, we must have $xy \in 0^+$, so $y \in 0^+$, that is $y = 0^k$ for some $k \geq 1$, so $xy^2 z = 0^{0+|y|} 1^p$ contains more 0s than 1s, so $xy^2 z \notin \mathcal{L}$.

Example 1.17. Consider that language $\mathcal{L} = \{ww \mid w \in \Sigma^*\}$ over the binary alphabet. Let $p \in \mathbb{N}$, Consider $w = 0^p 1 0^p 1$, of course $w \in \mathcal{L}$ and $|w| \geq p$ and any partition xyz with $|xy| \leq p$ and $|y| > 0$ satisfies $wy^i z = 0^{p+(i-1)|y|} 1 0^p \notin \mathcal{L}$ (for $i = 2$ this is $0^{p+|y|} 1 0^p 1$).

1.4.2 Charectarizing Regular Languages

Definition 1.20 (Indistinguishable by $\mathcal{L}$)**.** Given a language $\mathcal{L} \subset \Sigma^*$ define $R_\mathcal{L} \subset \Sigma^* \times \Sigma^*$, and for $x, y \in \Sigma^*$ we say $x \sim_\mathcal{L} y$ (that is, $(x, y) \in R_\mathcal{L}$)) if for any $z \in \Sigma^*$ it holds that $x \cdot z \in \mathcal{L} \iff y \cdot z \in \mathcal{L}$. We say that x, y are **indistinguishable** by $\mathcal{L}$.

[II] Can choose $j \leq p$ by taking the $p+1$ prefix of w

Example 1.18. Consider $\mathcal{L} = (0+1)^*0(0+1)$, and $11, 111 \in \mathcal{L}$, then $11 \sim 111$ since

$$11z \in \mathcal{L} \iff z \in \mathcal{L} \iff 111z \in \mathcal{L}$$

On the contrary, $10 \not\sim_{\mathcal{L}} 11$ since the word $z = 1$ is a distinguishing suffix

Claim 1.4.2. *$R_{\mathcal{L}}$ is an equivalence relation.*

Proof. By definition:

- Symmetric: Yes.
- Reflexive: Yes.
- Transitive: Let $x \sim y \sim w$. Assume $x \not\sim w$, so there exists $z \in \Sigma^*$ (WLOG) with $xz \in \mathcal{L}$ but $wz \notin \mathcal{L}$. Consider yz: if $yz \in \mathcal{L}$ this contradicts $y \sim w$. Otherwise - contradicts $x \sim y$.

□

Theorem 1.10 (Myhill Nerode)**.** *For $\mathcal{L}$ the following are equivalent:*

1. $\mathcal{L} \in REG$
2. $\left|\Sigma^*/\sim_{\mathcal{L}}\right| < \infty$

Proof. $(1 \Rightarrow 2)$ Let $\mathcal{A}$ be an automaton for $\mathcal{L}$, and define $\sim_{\mathcal{A}} \in \Sigma^* \times \Sigma^*$ as $x \sim_{\mathcal{A}} y$ if $\delta^*(q_0, x) = \delta^*(q_0, y)$ (that is - the run ends on the same state). Note that $\sim_{\mathcal{A}}$ is an equivalence relation [III].

(Claim.)**.** $x \sim_{\mathcal{A}} y \Rightarrow x \sim_{\mathcal{L}} y$

Proof (of the claim). If $\delta^*(q_0, x) = \delta^*(q_0, y)$ then for $z \in \Sigma^*$ we must have

$$\delta^*(q_0, xz) = \delta^*(\delta^*(q_0, x), z) = \delta^*(\delta^*(q_0, y), z) = \delta^*(q_0, yz)$$

□

By the claim - $\sim_{\mathcal{A}}$ refines[IV] $\sim_{\mathcal{L}}$, and of course $\left|\Sigma^*/\sim_{\mathcal{A}}\right| \leq |Q| < \infty$, thus so is $\left|\Sigma^*/\sim_{\mathcal{L}}\right|$.
$(2 \Rightarrow 1)$ Assume $\left|\Sigma^*/\sim_{\mathcal{L}}\right| < \infty$ and construct a DFA for $\mathcal{L}$. Denote an equivalence class by $[w]$. Then define:

$$Q = \Sigma^*/\sim_{\mathcal{L}}, \quad q_0 = [\varepsilon], \quad \delta([w], \sigma) = [w \cdot \sigma], \quad F = \{[w] \mid w \in \mathcal{L}\}$$

Note that δ is well defined, since for $w, w' \in [w]$ we must have $w\sigma \sim_{\mathcal{L}} w'\sigma$, since if z distinguishes $w\sigma, w'\sigma$ then σz distinguishes w, w'. A similar proof shows that F is well defined ("if not, then ε distinguishes").
Correctness of $\mathcal{A}$: for any word $w \in \Sigma^*$ we must have $\delta^*(q_0, w) = [w]$ (by induction on $|w|$). Thus, $w \in \mathcal{L}(\mathcal{A}) \iff w \in \mathcal{L}$, as required.

□

[III] Left as an exercise to the avid reader
[IV] That is - $\left|\Sigma^*/\sim_{\mathcal{A}}\right| \geq \left|\Sigma^*/\sim_{\mathcal{L}}\right|$

Example 1.19 (Of constructing such DFA)**.** Consider once again $\mathcal{L} = (0+1)^*0(0+1)$. The equivalence classes are:

$$[\varepsilon] = \varepsilon + 1 + (0+1)^*11$$
$$[0] = 0 + (0+1)^*10$$
$$[00] = (0+1)^*00$$
$$[01] = (0+1)^*01$$

And the automaton is:

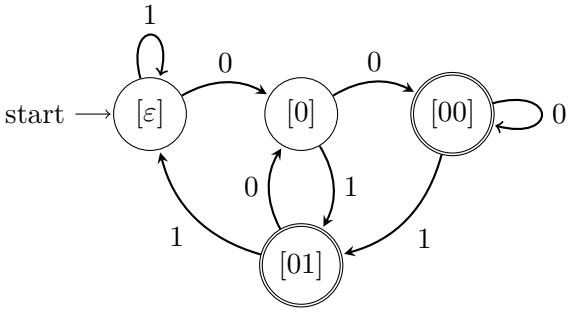


Figure 1.11: Automaton with equivalence classes as states

Example 1.20. Consider the following example (by Myhill Nerode) for language:

$$\mathcal{L} = \{0^i1^j \mid i,j \text{ are relative primes}^{\text{V}}\}$$

We show that this language is not regular:

(Claim.)**.** *For any p_1, p_2 prime numbers, $0^{p_1} \not\sim 0^{p_2}$.*

Proof. Since 1^{p_1} dinsinguishes them: $0^{p_1}1^{p_1} \notin \mathcal{L}$ but $0^{p_2}1^{p_1} \in \mathcal{L}$. □

1.4.3 Reduction of DFA to minimal DFA

Theorem 1.11 (Algorithm for reducing DFA to minimal DFA)**.** *Let $\mathcal{A}$ be a DFA for $\mathcal{L}$. Define a sequence of equivalence relations[a] $(\equiv_i)_{i=0} \subset Q \times Q$ such that:*

$$\forall i \quad q \equiv_i q' \iff \forall w \in \Sigma^{\leq i} \qquad \delta^*(q,w) \in F \iff \delta^*(q',w) \in F$$

Denote $\mathcal{A}^q$ to be $\mathcal{A}$ but with initial state q. The equivalence relation means that $\mathcal{L}(\mathcal{A}^q) = \mathcal{L}(\mathcal{A}^{q'}) \iff q \equiv_i q'$ for all i.

Computing $\equiv_i$:
$q \equiv_0 q' \iff (q \in F \iff q' \in F)$. Note that there are only 2 equivalence classes for $\equiv_0$. Assume we computed $\equiv_i$, and compute:
$q \equiv_{i+1} q'$ iff $q \equiv_i q'$ and for all $\sigma \in \Sigma$, $\delta(q,\sigma) \equiv_i \delta(q',\sigma)$. That is - given the relation table of $\equiv_i$, we can compute efficiently $\equiv_{i+1}$:

- *Computing $\equiv_0$ takes $O(n^2)$ with $|Q| = n$.*
- *Given $\equiv_i$, computing $\equiv_{i+1}$ takes $|\Sigma| \cdot O(n^2)$*

The process stops when $\equiv_i = \equiv_{i+1}$ (fixed point), we will show that there has to be a state like that: With every iteration - at least one equivalence class of $\equiv_i$ that partitioned into

[V] That is, $gcd(i,j) = 1$

more than on class in $\equiv_{i+1}$. Hence - the process converges with $O(n)$ iterations.

Putting it all together - the algorithm takes $O(|\Sigma|n^3)$.

[a]Proof omitted

Proof of correctness

Proposition. *If $\equiv_i = \equiv_{i+1}$ then $\equiv_j = \equiv_i$ for $j > i$.*

Proof. Inductively, consider $\equiv_{i+2}$:

$$q \equiv_{i+2} q' \iff q \equiv_{i+1} q' \quad \wedge \quad \forall\sigma \quad \delta(q,\sigma) \equiv_{i+1} \delta(q',\sigma) \iff q \equiv_i q'$$

□

Proposition. *Let $\mathcal{S} = \{S_1 \dots S_k\}$ the equivalence classes of $\equiv_*$ (the first fixed point). Consider $\mathcal{A} = (\mathcal{S}, \Sigma, [q_0], \delta', \mathcal{S}_F = \{S_i \mid S_i \subset F\})$, with $\delta'([q],\sigma) = [\delta(q,\sigma)]$. Everything is well defined. Then:*

$$q \equiv_i q' \iff \left(\mathcal{L}(\mathcal{A}^q) \cap \Sigma^{\leq i}\right) = \left(\mathcal{L}(\mathcal{A}^{q'}) \cap \Sigma^{\leq i}\right)$$

Proof. By induction over i:

(Base:). $q \equiv_0 q'$ *iff* $\varepsilon \in \mathcal{L}(\mathcal{A}^q) \iff \varepsilon \in \mathcal{L}(\mathcal{A}^{q'})$.

(Step:). ($\Rightarrow$) *By contradiction, assume* $\exists w \in \mathcal{L}(\mathcal{A}^q)$ *but* $w \notin \mathcal{L}(\mathcal{A}^{q'})$ *with* $|w| \leq i+1$. *Consider two cases:*

1. *If $|w| \leq i$ - contradiction to induction assumption.*
2. *If $|w| = i+1$, then $w = \sigma w'$. We know $w \in \mathcal{L}(\mathcal{A}^q)$, then:*

$$F \ni \delta^*(q, \sigma w') = \delta^*(\delta(q,\sigma), w)$$

Contradicting $\delta(q,\sigma) \equiv_i \delta(q',\sigma)$ since w' distinguishes them.

($\Leftarrow$) *By definition.*

□

Example 1.21. Consider the following automaton:

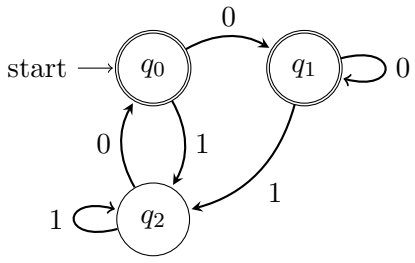


Figure 1.12: Automaton for $\varepsilon + (0+1)^* \cdot 0$

Then:

$$\equiv_0 = \{\{q_0, q_1\}, \{q_2\}\}$$

Now, check for $\equiv_1$: since $\delta(q_0,0) = \delta(q_2,0)$ and $\delta(q_0,1) = \delta(q_2,1)$,

$$\equiv_1 = \{\{q_0, q_1\}, \{q_2\}\}$$

Hence $\equiv_* = \equiv_0$, thus the minimal automaton:

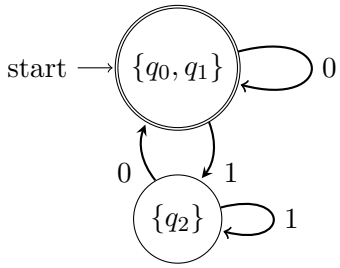


Figure 1.13: Minimal Automaton for $\varepsilon + (0+1)^* \cdot 0$

Definition 1.21 (Test this)**.** This is a definition

Chapter 2

Context Free Languages

Context Free Languages are another class of languages, like REG, denoted CFL. Any $\mathcal{L} \in$ CFL is defined by Context Free Grammar. The whole concept arrived from NLP and compilation.

2.1 Context Free Grammar

Intuitively, a **Context Free Grammar** consists of:

- A set of variables, eg $\{A, B\}$.
- A set of Terminals (similar to Alphabet), eg $\{0, 1, \#\}$
- Start Variables
- Substitution Rules

Formally:

Definition 2.1 (Context Free Grammar)**.** A **Context Free Grammar (CF Grammar)** is a 4-tuple:

$$\mathcal{G} = \langle V, \Sigma, R, S \rangle$$

With:

- V A finite set of variables.
- Σ A finite set of Terminals (similar to Alphabet).
- $S \in R$ A start variable
- R A finite set of Rules of the form $V \to (V \cup \Sigma)^*$

Definition 2.2 (Yields, Derives)**.** Let $\mathcal{G}$ be a CF grammar, and $u, v, w \in (V \cup \Sigma)^*$, and $A \to w \in R$, then we say uAv **Yields** uwv, and denote $uAv \Rightarrow uwv$.
let $u, v \in (V \cup \Sigma)^*$. We say that u **Derives** v, denote $u \Rightarrow^* v$ if there is a sequence of yields:

$$u = u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_m = v$$

Remark. For $m = 1$, we get u derives u.

Remark. Sometimes denote $\rightarrow$ instead of $\Rightarrow$.

Definition 2.3 (Language of CFG)**.** Let $\mathcal{G}$ be a CF grammar, we define **The Language of** $\mathcal{G}$ to be all the derivations from the start variable. That is:

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

Example 2.1. Consider the grammar $\mathcal{G}$:

$$A \rightarrow 0A1$$
$$A \rightarrow B$$
$$B \rightarrow \#$$

Then generate some words:

$$A \rightarrow 0A1 \rightarrow 00A11 \rightarrow 00B11 \rightarrow 00\#11$$
$$A \rightarrow B \rightarrow \#$$

Then $\mathcal{L}(\mathcal{G}) = \{0^n\#1^n \mid n \geq 0\}$

Example 2.2. Let $\mathcal{G} = \langle\{S, A\}, \{0, 1\}, R, S\rangle$ with R:

$$S \rightarrow A1A$$
$$A \rightarrow \varepsilon \mid 1A \mid 0A$$

With $\mid$ being the "or" operator. Think - what is $\mathcal{L}(\mathcal{G})$? $00 \notin \mathcal{L}(\mathcal{G})$ obviously. Now, notice that from A we can derive any $w \in \Sigma^*$, thus

$$\mathcal{L}(\mathcal{G}) = (0+1)^*1(0+1)^*$$

All words with at least one 1.

Example 2.3. Once again, $\mathcal{G}$ is CFG with rules:

$$S \longrightarrow aSb \mid SS \mid \varepsilon$$

Note that *abab* is in the language, consider the derivation tree:

$$\begin{array}{cccccc} & & & S & & \\ & & S & & S & \\ a & S & b & a & S & b \\ a & \varepsilon & b & a & \varepsilon & b \end{array}$$

In fact, $\mathcal{L}(\mathcal{G})$ is "valid parenthesis language".

Example 2.4. Consider $\mathcal{G}$ with:

$$S \longrightarrow aSa \mid bSb \mid \varepsilon$$

Claim 2.1.1. *$\mathcal{L}(\mathcal{G})$ are palindromes of even length over $\{a, b\}$.*

Proof. Pass. □

Definition 2.4 (Derivation Tree)**.** A **Derivation Tree of CFG** T is a rooted tree, with root S, nodes are variables in V, and leaves are terminals. The children of a node v are $u_1 \dots u_m$ with derivation rule $v \mapsto u_1 \dots u_m$

2.1.1 Closure Properties of CFL

Theorem 2.1 (Union)**.** *Let $\mathcal{L}_1, \mathcal{L}_2 \in CFL$, then $\mathcal{L}_1 \cup \mathcal{L}_2 \in CFL$.*

Proof. Let $\mathcal{G}_1, \mathcal{G}_2$ grammars for $\mathcal{L}_1, \mathcal{L}_2$, define$\mathcal{G}$ (WLOG $S \neq S_1, S_2$):

- $V = V_1 \sqcup V_2$
- $R = R_1 \cup R_2 \cup \{S \to S_1 \mid S_2\}$

Let $w \in \mathcal{L}_1 \cup \mathcal{L}_2$, WLOG $w \in \mathcal{L}_1$. Then $S \Rightarrow^*_{\mathcal{G}_1} w$, thus also $S \Rightarrow^*_{\mathcal{G}} w$ since $R_1 \subset R$.
Let $w \in \mathcal{L}(\mathcal{G})$, then $S \Rightarrow^*_{\mathcal{G}} w$. The first transition must be $S \to S_1$ or $S \to S_2$, WLOG $S \to S_1$. All following derivations must be from R_1 by $V_1 \cap V_2 = \emptyset$, hence taking the suffix of the derivation sequence yields a derivation sequence of w in $\mathcal{G}_1$, that is $w \in \mathcal{L}_1 \subset \mathcal{L}_1 \cup \mathcal{L}_2$. □

Theorem 2.2 (Concatenation)**.** *Let $\mathcal{L}_1, \mathcal{L}_2 \in CFL$, then $\mathcal{L}_1 \cdot \mathcal{L}_2 \in CFL$.*

Proof. Define $\mathcal{G} = \langle V_1 \sqcup V_2 \cup \{S\}, R_1 \cup R_2 \cup \{S \to S_1 S_2\}, S \rangle$. The proof is similar to the previous proof. □

Remark. CFL is **Not** closed under complement or intersection!

2.1.2 Chomsky Normal Form

All of our rules are $R \cup V \times (V \cup \Sigma^*)$. The "context-less" component is the fact that a rule $V \to uTw$ does not care what u, w are - hence, the context is meaningless.

Definition 2.5 (Chomsky Normal Form)**.** Let $\mathcal{G}$ be a CFG. We say that it is in **Chomsky Normal Form** if all of its derivation rules are of the form:

1. $S \to \varepsilon$
2. $A \in V$. $A \to BC$ with $B, C \neq S$
3. $A \to a \in \Sigma$.

Theorem 2.3 (Normalization)**.** *Any CFG $\mathcal{G}$ has an equivalent CFG $\mathcal{G}'$ with $\mathcal{G}'$ of Chomsky Normal Form.*

Proof. No formal proof, but we describe a normalization algorithm:

(Translation ($\mathcal{G}$))**.** 1. *Add S_0 a new start variable, and rule $S_0 \to S$.*

2. *Delete rules of the form $A \to \varepsilon$ for any $A \neq S_0$. For any such rule deleted, add rules:*

- *If there was a rule $R \to AB$, add rule $R \to B$.*
- *Generally - add any rule that substitutes any A in the output with ε (any combination). Note that we never add the rule that we deleted.*

3. *Delete terminals:*
   - *For any $\sigma \in \Sigma$, add a variable X_σ and rules $X_\sigma \to \sigma$.*
   - *Any occurance of σ in original derivation rules substitute with X_σ.*
4. *Delete rules of the form $A \to B$.*
   - *Delete these rules, and add new rules: Whenever A is in the output, substitute by B.*
5. *Delete long derivation $A \to V_1 \dots V_k$ for $k \geq 3$.*
   - *Delete those, and substitude them.*
   - *Define new variable $U_2 \dots U_{k-1}$, and add rules:*

$$A \to V_1U_2, U_2 \to V_2U_3, \dots, U_{k-1} \to V_{k-1}V_k$$

6. *Stop when no long rules remain.*

□

2.1.3 Stronger than REG

Theorem 2.4. *$REG \subset CFL$*

Proof. Let $\mathcal{L} \in$ REG, and $\mathcal{A}$ an automaton for $\mathcal{L}$. Define $\mathcal{G}_\mathcal{A}$ CFG the following way[I]:

- $V = \{V_q \mid q \in Q\}$
- $S = V_{q_0}$
- Rules: for any $q \in Q$ and $\sigma \in \Sigma$, add the rule:

$$V_q \to \sigma V_s \qquad s = \delta(q, \sigma)$$

And for any $q \in F$, add a rule

$$V_q \to \varepsilon$$

We show that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{G}_\mathcal{A})$. Consider $w = w_1 \dots w_m \in \mathcal{L}(\mathcal{A})$. Thus:

$$\begin{aligned}
&w \in \mathcal{L} \iff \\
&\text{the run } q_0 \dots q_m \text{ is accepting} \iff \\
&\forall i \quad \delta(q_i, w_{i+1})q_{i+1} \wedge q_m \in F \iff \\
&\text{The grammar contains the rules } V_{q_i} \to w_{i+1}V_{q_{i+1}} \wedge V_{q_m} \to \varepsilon \iff \\
&V_{q_0} \Rightarrow w_1V_{q_1} \Rightarrow \dots \Rightarrow wV_{q_m} \Rightarrow w\varepsilon = w \iff \\
&w \in \mathcal{L}(\mathcal{G}_\mathcal{A})
\end{aligned}$$

□

[I] This is called **Right Linear Language**, rules being $V \to \sigma V'$ or $V \to \varepsilon$

Example 2.5. Consider:

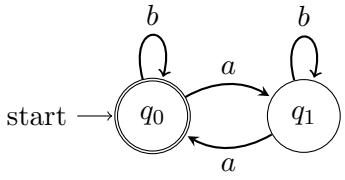


Figure 2.1: Automaton $\mathcal{A}$

With $\mathcal{L}$ being all words that have an even number of as. Then $\mathcal{G}_\mathcal{A}$ has the rules:

$$V_{q_0} \to aV_{q_1} \mid bV_{q_0} \mid \varepsilon$$
$$V_{q_1} \to aV_{q_0} \mid bV_{q_1}$$

Not all languages are CF

Example 2.6. Consider the language $\mathcal{L} = \{a^n b^n c^n \mid n \in \mathbb{N}\}$. This is not a context free language.

2.1.4 Ambiguity

Consider the grammar $\mathcal{G} = \langle \{E\}, \{+, \times, 0 \dots 9\}, R, E \rangle$ with rules:

$$E \to E + E \mid E \times E \mid 0 \dots 9$$

That is - this is the "calculator" language. The expression $3{+}5{\times}8$ has two derivation trees!

Definition 2.6 (Ambiguous Grammar)**.** A grammar $\mathcal{G}$ is **Ambiguous** if there is a word $w \in \mathcal{L}(\mathcal{G})$ with two derivation trees.

2.1.5 Non Context Free Languages

Just like REG, we have a necessary condition for a language to be CF, in the form of a pumping lemma.

Theorem 2.5 (Pumping Lemma for CFL)**.** *Let $\mathcal{L} \in CFL$. Then there exists $p \geq 1$ such that for any sufficiently long $w \in \mathcal{L}$ ($|w| \geq p$) there is a partitioning $w = uvxyz$ satisfying*

1. $|vy| > 0$
2. $|vxy| \leq p$
3. $\forall i \geq 0$, $uv^i xy^i z \in \mathcal{L}(\mathcal{G})$

Remark. Just like in the context of regular languages - we will use this lemma mostly to *refute* $\mathcal{L} \in$ CFL.

Example 2.7. Consider $\mathcal{L}_1 = \{a^n b^n c^m \mid n, m \geq 0\}$ and $\mathcal{L}_2 = \{a^n b^m c^m \mid n, m \geq 0\}$ context free languages. But $\{a^n b^n c^n \mid n \geq 0\} = \mathcal{L}_1 \cap \mathcal{L}_2$ is not context free (by the pumping lemma, shown later).

Proof. Let $\mathcal{G}$ be a CF grammar with $\mathcal{L}(\mathcal{G}) = \mathcal{L}$. Let b the maximal length of an output of derivation rule[II]. So $\deg(v) \leq b$ for any node of the derivation tree. Choose $p = b^{|V|+1}$. Let

[II] That is, the maximal $|\mathcal{W}|$ of $V \to \mathcal{W} \in (V \cup \Sigma)^*$.

$w \in \mathcal{L}$ such that $|w| \geq p$. Consider $\mathcal{T}_w$ a minimal derivation tree (in terms of height). Since $|w| \geq p = b^{|V|+1}$, $h(\mathcal{T}_w) \geq |V| + 1$, thus there is a path from the root to leaves with at least $|V| + 2$[III] nodes (since height counts edges). Thus in this path there is at least $|V| + 1$ nodes that are variables. In particular, there is a path with repetition of a variable R in the $|V| + 1$ nodes closest to the leaves. Consider the following partition (induced from $\mathcal{T}_w$) of $w = uvxyz$ with u, z is what is right/ left from R (respectively), v, y is the intermediate derivation from one R to another, and x is the middle (See figure below) . We must have:

1. $|vy| > 0$ since $\mathcal{T}_w$ is minimal. if both $v, y = \varepsilon$ then we used $R \overset{*}{\underset{\mathcal{G}}{\Longrightarrow}} R$, not minimal tree.
2. $|vxy| \leq p$ because the upper most height of the appearance of R is at most $|V|+1$ (consider some leaf arithmetic).
3. for $i = 0$, $S \overset{*}{\underset{\mathcal{G}}{\Longrightarrow}} uRz \overset{*}{\underset{\mathcal{G}}{\Longrightarrow}} uxz$ from the derivation tree, and for $i \geq 2$:

$$S \overset{*}{\Longrightarrow} uRz \overset{*}{\Longrightarrow} uvRyz \underset{i \text{ times}}{\overset{*}{\Longrightarrow}} uv^iRy^iz \overset{*}{\Longrightarrow} uv^ixy^iz \in \mathcal{L}(\mathcal{G})$$

□

Figure 2.2: Taken from the book

Example 2.8 (Application of pumping lemma)**.** Consider the language $\{a^nb^nc^n \mid n \geq 0\}$. Let $p \in \mathbb{N}$, and consider the word $w = a^pb^pc^p$. Of course $w \in \mathcal{L}$ and $|w| \geq 0$. Any partitioning $w = uvxyz$ satisfying (1) - (2). By the structure of w, $|vxy| \leq p$ implies $vxy \in a^*b^* + b^*c^*$. Therefore, consider $i = 2$, then $w_2 = uv^2xy^2z$ has more a from c, or more c than a, or more b than a, c. Either way - $w_2 \notin \mathcal{L}(\mathcal{G})$.

[III] This is important! we need $|V| + 1$ variables, and leaves are terminals.

Part II

Computability Theory

Chapter 3

Church-Turing Thesis

3.1 Turing Machines

Example 3.1 (Motivation for Turing Machines)**.** Consider the language $\mathcal{L} = \{w\#w \mid w \in (0+1)^*\}$, this is not CFL, in particular - not regular. With that said - given a word 00110#00110 (on the strip of a TM), one can *decide* if this word is in the language. This process can be described by a Turing machine:

- Check if the word is of the form $(0+1)^*\#(0+1)^*$. If not - *reject*.
- Start from the first state, go to the symmetrical opposite, *write* X on the corresponding letters. If some corresponding pair do not agree - *reject*.

Definition 3.1 (Deterministic Turing Machine)**.** $\mathcal{M} = \langle Q, \Sigma, \Gamma, q_0, q_{acc}, q_{rej}, \delta \rangle$ with:

- Q finite set of configurations.
- Σ alphabet, not containing **Blank Symbol** $\sqcup$.
- Γ is **Working Alphabet** (**Tape** Alphabet) - the things the TM can write on the strip. We demand $\Sigma \subset \Gamma$, and $\sqcup \in \Gamma$.
- $q_0, q_{acc} \neq q_{rej} \in Q$ are start, accepting and rejecting states respectively.
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$. Meaning - when the machine is in state q and the head is over a tape square containing a symbol γ, then $\delta(q, \gamma) = (q', \gamma', D)$ means that the machine overwrites γ with γ', goes to state q' and moves over the tape in direction D.

Remark. A DFA $\mathcal{A}$ is a specific instance of TM , the intuition is $\delta(q, \sigma) = (q', \sigma, R)$. We will not get into this here.

Remark. Note the difference between state and block on strip.

Remark. The Nondeterministic setting is simply $\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,R\}}$

Definition 3.2 (Configurations)**.** Let $\mathcal{M}$ be a TM , the **Configuration** in which the current tape content is the words uv (with $u, v \in \Gamma^*$), the head is over the first letter in v and state q - is denoted uqv.
The collection of all configurations is $\Gamma^* Q \Gamma^*$, the **Initial** configuration is $q_0 w$, an **Ac-**

cepting/ Rejecting configuration is one such that $q = q_{acc}, q_{rej}$ respectively. These two configurations are called **Halting** configurations.

Definition 3.3 (Yielding Configurations)**.** Let $a, b \in \Gamma, u, v \in \Gamma^*, q \in Q$ and consider the configuration $uaqbv$ (that is, current state is q and head over b on the strip). If $\delta(q, b) = (q', c, L)$, then we say $uaqbv$ **Yields** the configuration $uq'acv$, denoted:

$$uaqbv \longrightarrow uq'acv$$

We say that $uq'acv$ is consecutive (or subsequent) to $uaqbv$.
Similarly, if $\delta(q, b) = (q', c, R)$, we say $uaqbv$ **Yields** the configuration $uacq'v$, denoted

$$uaqbv \longrightarrow uacq'v$$

We say that $uacq'v$ is consecutive to $uaqbv$.

Definition 3.4 (Run of TM on a word)**.** Let $\mathcal{M}$ be a TM . **The Run** on $\mathcal{M}$ on $w \in \Sigma^*$ is a sequence $\mathcal{R} = C_0, C_1 \ldots$[a] such that C_0 is the initial configuration $C_0 = q_0 w$ and for any $i \geq 1$, C_{i+1} is **Yielded** by C_i, or that C_i is a halting configuration.

Definition 3.5 (Accepting Run)**.** We say the $w \in \Sigma^*$ is **accepted** by a TM $\mathcal{M}$ if there exists a sequence of configurations $C_0 \ldots C_m$[b] such that C_0 is the initial configuration of $\mathcal{M}$ on w - $q_0 w$, and for any $i < m$, C_{i+1} is consecutive of C_i, and C_m is accepting.

Definition 3.6 (Language of TM , Recognize Language)**.** We define the **Language of TM** $\mathcal{M}$:

$$\mathcal{L}(\mathcal{M}) = \{w \mid w \text{ is accepted by } \mathcal{M}\}$$

We say that $\mathcal{M}$ **Recognizes** $\mathcal{L}$ if $\mathcal{L}(\mathcal{M}) = \mathcal{L}$. Alternatively - $\mathcal{L}$ is **Turing-recognizable**.

Definition 3.7 (Recursively Enumerable Languages, coRE)**.** We define RE to be the class of all languages recognized by some Turing machine. Define $\mathcal{L} \in \text{coRE} \iff \mathcal{L}^c \in \text{RE}$

[a] Could be infinite!
[b] Note that m could be even smaller than $|w|$

Any TM has three options on an input w: Halting (and rejecting accepting) or to not halt at all. An important note here is that for a recognizable language, we only demand halting on words that are in $\mathcal{L}$; for other words - $\mathcal{M}$ may not even halt. Therefore, we prefer Turing machines that halt on all inputs.

Definition 3.8 (Decide a Language)**.** We say $\mathcal{M}$ **Decides** a language $\mathcal{L}$ if $\mathcal{L}(\mathcal{M}) = \mathcal{L}$ and $\mathcal{M}$ halts on all inputs.

Definition 3.9 (Recursive languages)**.** The class R is the class of all Turing decidable languages.

Remark. It is clear that R $\subset$ RE, but we do not yet know if the containment is proper.

Example 3.2. Consider the language $\mathcal{L} = \{w\#w \mid w \in (0+1)^*\}$. We want to show $\mathcal{L} \in$ R.

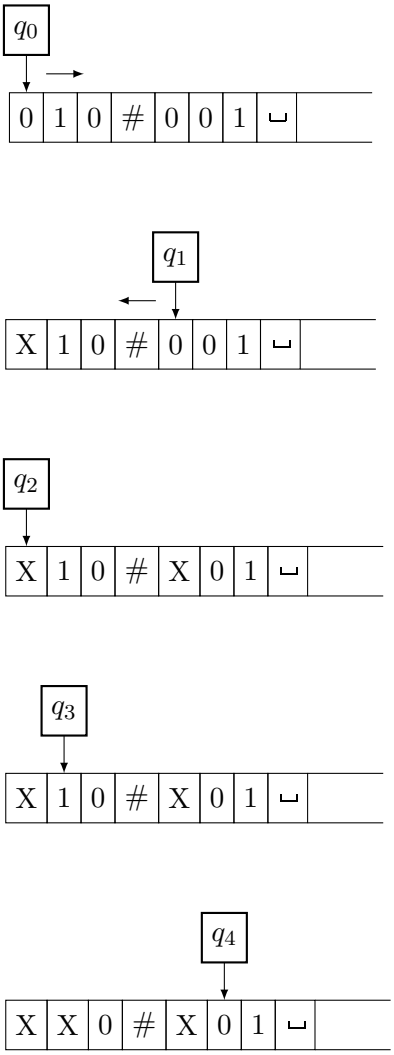


Figure 3.1: Example of the run on input 010#001 until transition to rejecting state

Idea:

1. Let the head be on the first block, if points on ␣ - reject.
2. If the current block is not #:
   - 2.1 Delete the current block, remember what was written.
   - 2.2 Move right until the last not-deleted character to the right of #.
   - 2.3 If this character is different from what we remembered, or is #, ␣ - reject. Else - delete it, and go left to the closest deleted character to #, and move one block right. Return to 2.
3. Go right to the first not deleted character to the right on #, if ␣ - accept. Else - reject.

As a state machine, this looks like this:

Figure 3.2: Taken From the Book

Definition 3.10 (coRE)**.** Define the class:

$$\text{coRE} = \{\mathcal{L} \mid \Sigma^* \setminus \mathcal{L} = \mathcal{L}^c \in \text{RE}\}$$

3.1.1 Closure Properties of R, RE

Theorem 3.1. *R is closed under complement.*

Proof. Let $\mathcal{L} \in$ R and $\mathcal{M}$ TM such that $\mathcal{L}(\mathcal{M}) = \mathcal{L}$. Define $\mathcal{M}'$ with $q'_{qcc} = q_{rej}$ and vice versa. Then still $\mathcal{M}'$ halts on any $w \in \Sigma^*$, and $w \in \mathcal{L}(\mathcal{M}')$ iff $w \notin \mathcal{L}(\mathcal{M}) = \mathcal{L}$ thus $\mathcal{L}(\mathcal{M}') = \mathcal{L}^c$. □

Theorem 3.2. *RE is closed under Union.*

Proof. Let $\mathcal{L}_1, \mathcal{L}_2 \in$ RE and $\mathcal{M}_1, \mathcal{M}_2$ appropriate TMs. Define $\mathcal{M}$ the following way: First - move the input of the strip one place to the right, and add \$ at the start (can be done, see recitation), and add one more \$ at the end of the word. This one will never be deleted. Leave space for a counter, and add one more \$ and a bit indicating in which TM we are at the moment, and one more \$. Afterwards - copy w after the final \$.

\$	w	\$	counter	\$	b	\$	w	

$\mathcal{M}$ emulates the current machine's (according to the bit) run for i (counter) steps. If the emulation accepts - accept the word. Otherwise - switch the bit. If $0 \to 1$, increase $i++$, check w again using the other TM .
Let $w \in \mathcal{L}_1 \cup \mathcal{L}_2$, then $\mathcal{M}_1$ (WLOG) accepts w after m steps, then when $\mathcal{M}$'s counter reaches m - $\mathcal{M}$ accepts w, thus $w \in \mathcal{L}(\mathcal{M})$. Conversely - if $w \in \mathcal{L}(\mathcal{M})$, then $\mathcal{M}$ reached some accepting state in some emulation - hence $w \in \mathcal{L}_1 \cup \mathcal{L}_2$. □

3.1.2 Relationship between Turing classes

Theorem 3.3.

$$coRE \cap RE = R$$

Proof. By definition - R $\subset$ RE. We show R $\subset$ coRE. Let $\mathcal{L} \in$ R and $\mathcal{M}_1$ a TM deciding $\mathcal{L}$. Then $\widetilde{\mathcal{M}}_1$ defined by $\widetilde{q}_{acc} = q_{rej}$[I] and vice versa, decides $\mathcal{L}^c$, hence $\mathcal{L} \in$ coRE, thus R $\subset$ RE $\cap$ coRE. Conversely - let $\mathcal{L} \in$ RE $\cap$ coRE, and $\mathcal{M}, \mathcal{M}^c$ TMs that recognize $\mathcal{L}, \mathcal{L}^c$ respectively. We build $\mathcal{M}$ that decides $\mathcal{L}$: Given $w \in \Sigma^*$, then $\mathcal{M}$ would operate in the following manner:

1. Initialize a counter $i = 0$.
2. While we did not stop:
   - 2.1 $i+ = 1$
   - 2.2 Run $\mathcal{M}, \mathcal{M}^c$ on w for i steps.
   - 2.3 If $\mathcal{M}$ accepted: Halt and accept.[II]

[I] Using the deterministic model. This in fact means that R is closed under negation.
[II] Can reject if rejects

2.4 If $\mathcal{M}^c$ accepted: Halt and reject[III]

It is guaranteed $w \in \mathcal{L} \sqcup \mathcal{L}^c$, thus $\exists i$ s.t one of $\mathcal{M}, \mathcal{M}^c$ halts and accepts after i steps - thus $\mathcal{M}$ always halts and either accepts or rejects and with the correct answer. □

3.2 Variants of TMs

3.2.1 Enumerators

An enumerator is a variant of Turing Machine - it is a TM with a "printer": It has no input. It writes a sequence of words separated by # on the output strip. The language of an enumerator $\mathcal{E}$ is defined:

$$\mathcal{L}(\mathcal{E}) = \{w \mid \mathcal{E} \text{ prints } w \text{ at least once}\}$$

Theorem 3.4. *$\mathcal{L} \in RE \iff$ there exists an enumerator $\mathcal{E}$ s.t $\mathcal{L}(\mathcal{E}) = \mathcal{L}$.*

Proof. Let $\mathcal{E}$ be an enumerator with $\mathcal{L}(\mathcal{E}) = \mathcal{L}$. Define $\mathcal{M}$: given $w \in \Sigma^*$, $\mathcal{M}$ runs $\mathcal{E}$ and whenever $\mathcal{E}$ prints a new word y, $\mathcal{M}$ checks if $w = y$. If so - $\mathcal{M}$ halts and accepts. Correctness is obvious.
Let $\mathcal{M}$ recognize $\mathcal{L}$, and build $\mathcal{E}$ s.t $\mathcal{L}(\mathcal{E}) = \mathcal{E}$. Let $\mathcal{T}$ be a lexicographic order of Σ^*[IV]. $\mathcal{E}$ would operate the following way: Initialize a counter i, run $\mathcal{M}$ on $w_1 \dots w_i$ for i steps and print whenever $\mathcal{M}$ accepts w_j. Note that this always halts after at most i^2 steps. Afterwards - $i+ = 1$. Once again - correctness is obvious. □

3.2.2 Nondeterminism

Definition 3.11 (nondeterministic TM)**.** A **Nondeterministic TM** is defined in the expected way - by allowing "guesses". This is captured ny the fact that the transition function is:

$$\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{R,L\}}$$

Definition 3.12 (Decider)**.** We say that a nondeterministic TM $\mathcal{M}$ is a **decider** if all computations of $\mathcal{M}$ halt.

Definition 3.13 (Nondeterministic runtime)**.** The runtime of a nondeterministic decider $\mathcal{M}$ on a word w is the longest path in the run tree of $\mathcal{M}(w)$.

Example 3.3. Consider $\mathcal{C} = \{\langle n \rangle \mid n \text{ is not a prime number}\}$. We build a nondeterministic TM that decides $\mathcal{C}$: Given $\langle n \rangle$, guess a prime $p \leq n$ and check if it is a divisor of n. If so - accept, otherwise reject.

Theorem 3.5 (Equivalence)**.** *For any nodeterministic decider $\mathcal{N}$ there exists a deterministic TM $\mathcal{M}$ with $\mathcal{L}(\mathcal{N}) = \mathcal{L}(\mathcal{M})$.*

[III] Can accept if rejects
[IV] Countable!

Proof. The intuition here is using BFS over the decision tree of $\mathcal{N}$: The root is the initial configuration and for any node, its children are all possible consecutive configurations. Define $\mathcal{M}$ the following way:
Initialize an iteration. In the ith iteration - simulate $\mathcal{N}$'s run up to depth i in it's run tree of $\mathcal{N}$ (this is the BFS idea). Note that the splitting degree of the tree is bounded by some constant k - hence BFS can be done. If at some point $\mathcal{N}$ accepts - accept, otherwise - reject (since $\mathcal{N}$ is a decider, this is well defined). Therefore

$$\mathcal{M}(w) = acc \iff \text{There exists an accepting branch in } \mathcal{N}\text{'s run tree} \iff w \in \mathcal{L}(\mathcal{N})$$

□

What is the **cost** of translation? If $\mathcal{N}$ runs t steps on w, then $\mathcal{M}$ runs $O(k^t \cdot t^2) = O(2^t)$ steps, with k^t the maximal degree of a node in the run tree, and t^2 comes from BFS (number of edges).

Theorem 3.6 (Characterization of "nondeterministic languages")**.** *Let $\mathcal{L} \subset \Sigma^*$. There exists a nondeterministic $\mathcal{N}$ with $\mathcal{L} = \mathcal{L}(\mathcal{N})$ if and only if there exists $\mathcal{K} \in R$ such that*

$$\mathcal{L} = \{x \in \Sigma^* \mid \exists y \quad x\#y \in \mathcal{K}\}$$

We should think of y as a "witness" of $x \in \mathcal{L}$ - that is, y is an "encoding of an accepting run of $\mathcal{N}$ over x.

Remark. A machine $\mathcal{T}$ that decides $\mathcal{K}$ is called a **verifier** for $\mathcal{L}$.

Proof. Let $\mathcal{N}$ be s.t $\mathcal{L}(\mathcal{N}) = \mathcal{L}$, and denote

$$\mathcal{K} = \{x\#y \mid y \text{ describes an accepting run of } \mathcal{N} \text{ over } x\}$$

Then if $x \in \mathcal{L}$ there exists an accepting run y of $\mathcal{N}$ over x, hence $x\#y \in \mathcal{K}$. Conversely - if $x\#y \in \mathcal{K}$ then there exists an accepting run of $\mathcal{N}$ over x (described by y) - so the construction is correct. It's left to show that $\mathcal{K} \in$ R: Let $\mathcal{T}$ be a deterministic TM that receives $x\#y$ and checks that y is a valid description of a run of $\mathcal{N}$ over x, and that the first configuration of y corresponds to $q^{\mathcal{N}}_{acc}$, that all configurations are subsequent and if the last configuration is $q^{\mathcal{N}}_{acc}$. If so - accept. Otherwise, reject. Thus $\mathcal{K} \in$ R.

Conversely - Assume that there exists $\mathcal{K} \in$ R and $\mathcal{L}$ is of the ugly form. Construct $\mathcal{N}$ the following way: Guess all possible $\#y \in \Sigma^*_{\mathcal{K}}$. There are infinitely many possible $\#y$ - so the guess is done by sequentially guessing if adding another letter from $\Sigma_{\mathcal{K}}$ or to halt - this is of splitting degree $|\Sigma_{\mathcal{K}}| + 1$. After the guess - run $\mathcal{T}_{\mathcal{K}}$ over $x\#y$ (that is deterministic!). As for correctness - $x \in \mathcal{L}(\mathcal{N})$ iff there is some $\#y$ such that $x\#y \in \mathcal{K}$. Note that $\mathcal{N}$ doesn't necessarily halt over all x. □

Example 3.4. Consider $\mathcal{C}$ from earlier. Then $\mathcal{K}$ should be thought of as $x\#\langle p \text{ devisor}\rangle$: $\mathcal{T}_{\mathcal{K}}$ over $x\#y$ will check $y \overset{?}{\mid} x$.

3.3 The Definition of an Algorithm

They developed the concept independently. Turing defined Turing Machine, and Church showed that λ-calculus is equivalent to TM .
In 8.8.1900 - the best mathematicians met and described Hilbert's 23 problems. The 10'th problem was the following:

Describe an algorithm that given a polynomial $p \in \mathbb{Z}[x_1 \dots x_k]$ - decide if there is an integer root $(n_1 \dots n_k) \in \mathbb{Z}^k$

As for the definition of algorithm - Hilbert described an algorithm as "A process that can be decided after a finite number of steps". Today (1970), we know that there is no solution: This problem is not in R, thus this problem is undecidable. For our needs - an algorithm can be described in one of three levels:

1. (Formal description) A Turing Machine $\mathcal{M}$
2. (Implementation description) Description of operation of $\mathcal{M}$
3. (High-Level description)Description by Pseudo-Code.

We would like to be convinced that $2 \iff 3$. There is a gap in **terminology**[V]. We will denote $\langle A \rangle$ to describe an instance A. For example - describing Hilbert's problem as:

$$D = \{\langle p \rangle \mid p \text{ has an integer root}\}$$

This is thinking of polynomials as words in some language, and a polynomial $6x^3yz^2 + 3xy^2 - x^3 - 10$ is described on a strip as:

$	6	x	3	y	z	2	+	3	x	y	2	−	x	3	−	10	␣	⋯	

Figure 3.3: Every number is encoded in binary, and we add some separators to avoid ambiguity

Example 3.5. Describe a TM for the problem of G connectness. Formally:

$$\mathcal{L} = \{\langle G \rangle \mid G \text{ is an undirected connected graph}\}$$

$G = (V, E)$ is encoded by enumerating the nodes $1 \dots n$ and writing them down in binary, separated by #. When the nodes have ended - write \$ and start encoding the edges by pairs of nodes separated by # (or a new symbol, whatever)

$	0	0	#	0	1	#	1	0	$	0	0	#	0	1	$	0	1	#	␣	␣	␣	␣	␣	␣	⋯	

Note that $\mathcal{L}^c = \{w \mid \text{does not describe a graph, or describes an unconnected graph}\}$. We will describe an algorithm to implement the algorithm:
$\Gamma = \{0, 1, \#, \$, 0_T, 1_T, 0_A, 1_A, 0_C, 1_C\}$ with the convention that a node with the first letter in its encoding is $0_T, 1_T$ is in T, $0_C, 1_C$ in C and $1_A, 0_A$ is active (currently tested).

1. T-mark the first node.
2. While there are T-marked nodes:
   - 2.1 A-mark a T-marked node.
   - 2.2 Scan the node list. If there is an unmarked node - check if there is an edge between it and the A-marked node. If so - T-mark it.
   - 2.3 C-mark the previously A-marked node.
3. If all nodes are C-marked, accept. Else - reject.

The goal here is to generally understand that Turing machines describe algorithms

[V] Graphs, matrices, etc - compared to words.

Chapter 4

Decidability and Reducability

In this chapter - we explore the question of decidability. Namely - what languages are decidable? Are there languages $\in$ R but not in RE? How rich is RE?
Let Σ be a finite alphabet. Since there are 2^{Σ^*} languages, but all Turing machines are finite sequences - thus are countable. Since there is a surjective function TM $\rightarrow$ RE, RE is countable. but RE $\subset 2^{\Sigma^*}$ - so there must be at least one language with no TM - hence RE is not everything.

4.1 Undecidibility

Theorem 4.1 (Exsistance of undecidable langauges)**.**

$$\exists \mathcal{L} \notin R$$

In particular - there exists a language $\mathcal{L} \notin RE$ or $\mathcal{L}^c \notin coRE$.

Proof. Consider the language $A_{\text{TM}} = \{\langle \mathcal{M}, w\rangle \mid \mathcal{M} \text{ accepts } w\}$. $A_{\text{TM}} \in$ RE for a TM $\mathcal{H}$ that recognizes A_{TM} , given $\langle \mathcal{M}, w\rangle$ will run $\mathcal{M}$ on w.

Claim 4.1.1. *$A_{TM} \notin R$.*

By contradiction - let $\mathcal{H}$ be a TM that decides A_{TM} , that is:

$$\mathcal{H}\left(\langle \mathcal{M}, w\rangle\right) = \begin{cases} acc & \mathcal{M}(w) = acc \\ rej & \mathcal{M}(w) \neq acc \end{cases}$$

Note that the second cases encapsulates two options: either $\mathcal{M}(w)$ halts and rejects, or does not even stop! Now we build a new TM $\mathcal{D}$ that gets an input $\langle \mathcal{M}\rangle$ and operates the following way:

$$\mathcal{D}(\langle \mathcal{M}\rangle) = \begin{cases} acc & \mathcal{M}(\langle \mathcal{M}\rangle) = acc \\ rej & \mathcal{M}(\langle \mathcal{M}\rangle) \neq acc \end{cases}$$

Now, given $\mathcal{D}$ - build $\widetilde{\mathcal{D}}$ by flipping q_{acc} and q_{rej}.[I]
Consider $\widetilde{\mathcal{D}}\left(\left\langle \widetilde{\mathcal{D}}\right\rangle\right)$. That is:

$$\widetilde{\mathcal{D}}\left(\left\langle \widetilde{\mathcal{D}}\right\rangle\right) = \begin{cases} acc & \widetilde{\mathcal{D}}\left(\left\langle \widetilde{\mathcal{D}}\right\rangle\right) \neq acc \\ rej & \widetilde{\mathcal{D}}\left(\left\langle \widetilde{\mathcal{D}}\right\rangle\right) = acc \end{cases}$$

Which is a contradiction. □

[I] This is called **Diagonalization**.

4.1.1 The Halting Problem

Definition 4.1. Define $\text{HALT}_{\text{TM}} = \{\langle M, w\rangle \mid M \text{ halts on } w\}$, that is - all Turing machines that halt on a word w.
Also define $\text{HALT}^{\varepsilon}_{\text{TM}} = \{\langle \mathcal{M}\rangle \mid \mathcal{M}(\varepsilon) \text{ halts}\}$

Theorem 4.2. *$HALT_{TM} \notin R$*

Proof. Let $\mathcal{M}'$ that decides HALT_{TM} , then if $\langle \mathcal{M}, w\rangle \notin HALT$, then $\mathcal{M}$ doesn't halt on w, so $\langle \mathcal{M}, w\rangle \notin A_{\text{TM}}$, and the machine $\langle \mathcal{M}, w\rangle$ decides A_{TM} . Otherwise - if $\langle \mathcal{M}, w\rangle \in \text{HALT}_{\text{TM}}$, then we can run $\mathcal{M}$ on w and decide accordingly. That is - deciding $HALT$ means deciding A_{TM} - and as such, $\text{HALT}_{\text{TM}} \notin \text{R}$ □

This is called a proof by **Reduction**,

4.2 Reductions

Definition 4.2 (Computable Function)**.** We say $f : \Sigma^* \to \Sigma^*$ is **Computable** if there exists $\mathcal{M}_f$ that on w halts with $f(w)$ on the strip.

Example 4.1. $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ by $(x, y) \mapsto x + y$ by their unary representation. Then as input:

1	1	1	1	#	1	1	1	␣	␣	␣	␣	␣	␣	␣	␣	···	

and as output

1	1	1	1	1	1	1	␣	␣	␣	␣	␣	␣	␣	␣	␣	···	

Example 4.2. A function $f : \text{TM} \to \text{TM}$ such that $f(\mathcal{M})$ is a TM that does not halt on $w \notin \mathcal{L}(\mathcal{M})$. That is - the machine $f(\mathcal{M})$ loops whenever $\mathcal{M}$ rejects. THis is computable - we'll add a state q_{loop} and transitions that were suppose to end with q_{rej} will now go to q_{loop}, and of course a loop $q_{loop} \to q_{loop}$. Correctness is obvious.

Example 4.3 (Uncomputable function - BB)**.** Let $\Gamma = \{0, 1, ␣\}$. Define BB^{II} by

$BB(k) =$ The maximal number of 1's written on the strip of a TM with k states over Γ that halt over ε

Proposition (1)**.** *$BB(k)$ is strictly increasing.*

Proof. Let $k \in \mathbb{N}$ and $\mathcal{M}$ that achieve the maximum (that is - $\mathcal{M}$ of k states and there are $BB(k)$ ones after $\mathcal{M}(\varepsilon)$ halts). Construct $\mathcal{M}'$ by switching $q^{\mathcal{M}}_{acc}$ by a new state that searches for a blank space on the strips and writes down 1 (then moves to an accepting state). Thus $BB(k+1) \geq BB(k) + 1 > BB(k)$. □

Proposition (2)**.** *BB is not computable. That is, $BB \notin R$.*

[II] Busy Beaver

Proof. By contradiction - let $\mathcal{H}$ be a TM that computes BB, and assume $\mathcal{H}$ has n states. WLOG $\mathcal{H}$ accepts as input and outputs numbers in binary representation. First - construct $\mathcal{M}_n$ with $\lceil \log(n) \rceil + 1$ states that halts over ε and returns the binary representation of $2n$. Define a TM $\mathcal{M}$ that given ε, run $\mathcal{M}_n$ and then run $\mathcal{H}$ (over $2n$ in binary representation - now written on the strip). That is - $\mathcal{M}$ computes $BB(2n)$. Now - $\mathcal{M}$ has $n + \lceil \log(n) \rceil + 1$ states, and computes $BB(2n)$ - but $2n > n + \lceil \log(n) \rceil + 1$. This means that $\mathcal{M}$ shows that $BB(2n) \leq BB(n + \lceil \log(n) \rceil + 1)$ - which is a contradiction to proposition (1). □

Definition 4.3 (Mapping Reducable)**.** We say that $A \subset \Sigma^*$ is **Mapping Reducable** to $B \subset \Sigma^*$ if there exists a computable function $f : \Sigma^* \to \Sigma^*$ such that $w \in A \iff f(w) \in B$.
The function f is called **Reduction** from A to B.
It is common to denote $A \leq_m B$.

Remark. This allows us to translate membership questions.

Example 4.4. Consider $A = \{x \mid |x| \leq 5\}, B = \{x \mid |x| \leq 10\}$ over $\mathbb{Z}$. Note that the identity function is *not* a reduction. $f(x) = 2x$ is a reduction.

Theorem 4.3 ("Reduction theorem")**.** *If A is reducable to B, then $B \in R \Rightarrow A \in R$.*

Remark. It is better to think of the theorem in the counter-positive description.

Proof. Let $B \in$ R, and let $\mathcal{M}_B, f$ be a TM and a reduction from A to B. Define $\mathcal{M}_A$ by the following procedure, given w:

1. Compute $M_{f(w)}$
2. Run $\mathcal{M}_B(f(w))$.

It is guaranteed that $\mathcal{M}_B$ accepts $M_{f(w)}$ iff $f(w) \in B$ iff $w \in A$, and $\mathcal{M}_A$ halts since $\mathcal{M}_f, \mathcal{M}_B$ halts. □

Remark. We will use this theorem mostly to show Undecidability. As usual - use the counter-positive of the theorem: $A \leq_m B$ and $A \notin$ R, then $B \notin$ R.

Theorem 4.4 (More variations on Reduction theorem)**.** *Let $\mathcal{L}_1, \mathcal{L}_2$ languages over Σ.*

1. $\mathcal{L}_2 \in RE \Longrightarrow \mathcal{L}_1 \in RE$
2. $\mathcal{L}_2 \in coRE \Longrightarrow \mathcal{L}_1 \in coRE$

Proof (of 2.): Let $\mathcal{L}_1 \notin$ coRE, and $f : \Sigma^* \to \Sigma^*$ a reduction $\mathcal{L}_1 \leq_m \mathcal{L}_2$, that is $f(x) \in \mathcal{L}_2 \iff x \in \mathcal{L}_1$. Notice that f is therefore a reduction $\mathcal{L}_1^c \leq_m \mathcal{L}_2^c$. That is - $\mathcal{L}_1^c \notin$ RE implies $\mathcal{L}_2^c \notin$ RE, hence $\mathcal{L}_2 \notin$ coRE. □

Example 4.5. (Another way to show the same thing) Consider HALT_{TM} , we want to show $A_{\text{TM}} \leq_m \text{HALT}_{\text{TM}}$, thus $\text{HALT}_{\text{TM}} \notin$ R.
Consider the function $f : \langle \mathcal{M}, w \rangle \mapsto \langle \mathcal{M}', w' \rangle$ Given $\langle \mathcal{M}, w \rangle$, create (the machine $\mathcal{M}_f$ will create) $\langle \mathcal{M}', w' \rangle$ such that $\langle \mathcal{M}, w \rangle \in A_{\text{TM}} \iff \langle \mathcal{M}', w' \rangle \in \text{HALT}_{\text{TM}}$. Consider the same function from before - f would be the function that transforms $\mathcal{M}$ into $\mathcal{M}'$ that halts only if a word is accepted. Note that indeed - if $\langle \mathcal{M}, w \rangle \in A_{\text{TM}}$, and as such $\langle \mathcal{M}', w' \rangle \in \text{HALT}_{\text{TM}}$. If $\langle \mathcal{M}, w \rangle \notin A_{\text{TM}}$ then $\mathcal{M}$ doesn't accept w, as such - $\mathcal{M}'$ doesn't halt on w, so $\langle \mathcal{M}', w' \rangle \notin \text{HALT}_{\text{TM}}$, since we are not guaranteed that $\mathcal{M}'$ halts. Hence f is a reduction, thus $\text{HALT}_{\text{TM}} \notin$ R.

Example 4.6. Consider $\text{HALT}^{\varepsilon}_{\text{TM}} \in \text{RE}$, and $f : \text{HALT}_{\text{TM}} \to \text{HALT}^{\varepsilon}_{\text{TM}}$ defined by

$\langle \mathcal{M}, w \rangle \overset{f}{\mapsto} \mathcal{M}_w$ $\mathcal{M}_w$ Moves the reading head to the left, write w on the strip and runs $\mathcal{M}$ on w

Of course $\mathcal{M}(w)$ halts iff $\mathcal{M}_w$ halts on ε, thus f is a mapping reduction from $\text{HALT}^{\varepsilon}_{\text{TM}}$ to HALT_{TM} , thus $\text{HALT}^{\varepsilon}_{\text{TM}} \notin \text{R}$.

Note that $g : \text{HALT}^{\varepsilon}_{\text{TM}} \to \text{HALT}_{\text{TM}}$ by $\langle \mathcal{M} \rangle \overset{g}{\mapsto} \langle \mathcal{M}, \varepsilon \rangle$ is a reduction.

Example 4.7 (More sophisticated)**.** Consider the language REG_{TM} of all TM such that $\mathcal{L}(\mathcal{M}) \in \text{REG}$. We show $\text{REG}_{\text{TM}} \notin \text{R}$. We show a reduction $A_{\text{TM}} \leq_m \text{REG}_{\text{TM}}$:

$$f : A_{\text{TM}} \to \text{REG}_{\text{TM}} \quad \langle \mathcal{M}, w \rangle \mapsto \langle \mathcal{M}' \rangle$$

$\mathcal{M}'$ operates over $x \in \{0,1\}^*$ the following way:
If $x \in \{0^n 1^n\}$ then $\mathcal{M}'(x) = acc$, otherwise - $\mathcal{M}'$ runs $\mathcal{M}$ on w and behaves the same way. Note that the computation of $f(\langle \mathcal{M}, w \rangle)$ is within finite time. We show $\langle \mathcal{M}, w \rangle \in A_{\text{TM}} \iff \mathcal{M}' \in \text{REG}_{\text{TM}}$:
If $\mathcal{M}(w) = acc$, then $\mathcal{L}(\mathcal{M}') = (0+1)^* \in \text{REG}$, thus $\langle \mathcal{M}' \rangle \in \text{REG}_{\text{TM}}$. If $\mathcal{M}(w) = rej$, then $\mathcal{L}(\mathcal{M}') = \{0^n 1^n\} \notin \text{REG}$, thus $\langle \mathcal{M}' \rangle \notin \text{REG}_{\text{TM}}$ - hence f is indeed a reduction $\text{REG}_{\text{TM}} \to A_{\text{TM}}$.

Example 4.8. Let $\text{ALL}_{\text{TM}} = \{\langle \mathcal{M} \rangle \mid \mathcal{L}(\mathcal{M}) = \Sigma^*\}$, we show that $\text{ALL}_{\text{TM}} \notin \text{RE} \cup \text{coRE}$:
We show $\text{ALL}_{\text{TM}} \notin \text{coRE}$ by reduction from A_{TM} to ALL_{TM} : We want a function $\langle \mathcal{M}, w \rangle \overset{f}{\mapsto} \mathcal{M}'$ with $\mathcal{M}(w) = acc$ iff $\mathcal{L}(\mathcal{M}') = \Sigma^*$. Define $\mathcal{M}'$ to operate the following way: On input x, accept iff $\mathcal{M}(w)$ accepts, that is $\mathcal{M}'(x) = \mathcal{M}(w)$ for any $x \in \Sigma^*$. Note that $\mathcal{L}(\mathcal{M}') = \Sigma^*$ iff $\mathcal{M}'$ accepts all x iff $\mathcal{M}(w) = acc$, that is f is indeed a mapping reduction $A_{\text{TM}} \leq_m \text{ALL}_{\text{TM}}$. Since $A_{\text{TM}} \notin \text{coRE}$, so is ALL_{TM} .
Now, we show $\text{ALL}_{\text{TM}} \notin \text{RE}$. We know that $A^c_{\text{TM}} \notin \text{RE}$, hence define

$$f : A^c_{\text{TM}} \to \text{ALL}_{\text{TM}} \quad \langle \mathcal{M}, w \rangle \mapsto \mathcal{T}_{\mathcal{M},w} = \mathcal{T}$$

Given x, $\mathcal{T}$ would emulate $\mathcal{M}(w)$ for $|x|$ steps, and reject iff $\mathcal{M}$ accepted w. If $\langle \mathcal{M}, w \rangle \notin A_{\text{TM}}$, then $\mathcal{M}(w) \neq acc$, then $\mathcal{T}$ would accept any word x, hence $\mathcal{L}(\mathcal{T}) = \Sigma^*$. Otherwise - for a large enough x, $\mathcal{T}(x) = rej$, hence $\mathcal{L}(\mathcal{T}) \neq \Sigma^*$. Therefore $A^c_{\text{TM}} \leq_m \text{ALL}_{\text{TM}}$. Since $A^c_{\text{TM}} \notin \text{RE}$, so is ALL_{TM} .

> There are many more examples in Recitation 7 notes - one should go over them. Reductions are difficult and are learned by examples and exercises.

Then Tiling Problem

The Tiling problem is an algorithmic problem. The input is a finite set T of tiles with an initial tile t_{init}. Any tile is a square (triangalized) with different colors, and a legal tiling is such that two tiles are possible neighbors if the colors match. Formally - there are horizontal relations $H \subset T \times T$ and vertical relations $V \subset T \times T$. A legal $n \times n$ tiling is a function $f : [n] \times [n] \to T$ such that:

- $f(1,1) = t_{init}$
- For any i, j: $H\left(f(i,j), f(i+1,j)\right)$ and $V\left(f(i,j), f(i,j+1)\right)$

Formalizing with our tools

Let $TILE = \{\langle T, H, V, t_{init}\rangle \mid \text{There is a legal tiling for all } n\}$. We would like to decide where does $TILE$ reside in the language classes.

Proposition. *$TILE \in coRE$.*

Proof. Consider T that recognizes $TILE^c$:

1. Initialize $i = 1$
2. Check all tilings of $[i] \times [i]$.
   - If found a legal tiling - $i+ = 1$.
   - If no legal tiling - accept.

□

Proposition. *$TILE \notin RE$.*

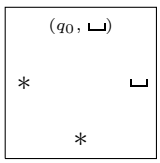
Proof. Surprisingly - we reduce $(HALT^{\varepsilon}_{\text{TM}})^c$ to $TILE$. The intuition is that an infinite run of $\mathcal{M}$ on ε corresponds to a tiling of $\mathbb{N} \times \mathbb{N}$.

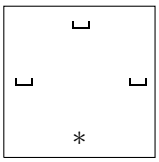
Claim 4.2.1. *There is a legal tiling for all n $\iff$ there is a legal tiling of $\mathbb{N} \times \mathbb{N}$.*

Proof (of the claim). $\Leftarrow$ is easy. We prove $\Rightarrow$. Recall KÅŚnig's lemma[III]. We construct a tree with i^{th} level nodes corresponding to $[i] \times [i]$ tilings, and an edge between a subtiling $f_{i-1} \subset f_i$. This tree is indeed connected and acyclic [IV]. Thus there is an infinite path - corresponding to a tiling of $\mathbb{N} \times \mathbb{N}$. □

We now construct the reduction. The idea - every floor in the tiling is a configuration, and transition between floors correspond to transition between subsequent configuration. We need to define T so that an infinite tiling corresponds to a run:

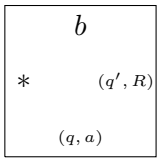
- The tiles in the first row:

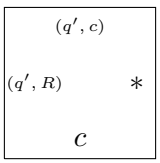




(a) Initial tile

- Tiles for simulating R movement: For any transition $\delta(q, a) = (q', b, R)$ for $q \neq q_{acc}, q_{rej}$ we add $|\Gamma| + 1$ tiles

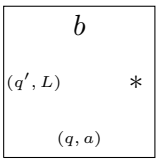


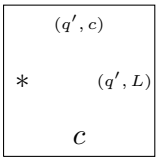


(a) for all $c \in \Gamma$

[III] In an infinite tree with a finite degree for any node there is an infinite path.
[IV] Add explanation

- Tiles for simulating L movement: For any transition $\delta(q,a) = (q',b,L)$ for $q \neq q_{acc}, q_{rej}$ we add $|\Gamma| + 1$ tiles





(a) for all $c \in \Gamma$

- Padding:

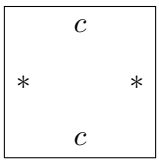


Figure 4.4: for all $c \in \Gamma$

Note that this defines a mapping from TMs $\langle \mathcal{M} \rangle$ into legal encodings of $TILE$. It is left to show that this is computable and that the reduction is correct:
For computability - given an encoding of δ, there are finitely many tiles to encode - so this is computable.
For correctness - there exists an infinite tiling of our construction iff there is an inifnite sequence of configurations $C_0 \ldots$ such that C_i is written in the transition from row i to row $i+1$, iff C_0 an initial configuration of $\mathcal{M}$ over ε and C_{i+1} is subsequent to C_i iff $\mathcal{M}(\varepsilon)$ doesn't halt. □

Part III

Complexity Theory

Chapter 5

Time Complexity

5.1 Measuring Complexity

Definition 5.1 (TIME)**.** For a function $t : \mathbb{N} \to \mathbb{N}$ we define

$$\text{TIME}(t(n)) = \{\mathcal{L} \mid \exists \mathcal{M} \text{ that decides } \mathcal{L} \text{ for } |w| = n, \text{ halts after } O(t(n)) \text{ steps}\}$$

Example 5.1. For $\mathcal{L}_{eq} = \{0^n 1^n\}$ we saw $\mathcal{L}_{eq} \in \text{TIME}(n^2)$. In fact - $\mathcal{L}_{eq} \in \text{TIME}(n \log n)$[I]. A TM with two strips can decide $\mathcal{L}_{eq}$ in $O(n)$ steps: Copy the 0 prefix to the second strip and compare. Even though - $\mathcal{L}_{eq} \notin \text{TIME}(n)$.[II]

Theorem 5.1. *Let $t(n)$ be a function that satisfies $t(n) \geq n$[a], then for any multi-strip TM that runs in $t(n)$ time, there is an equivalent single strip TM that runs in $t^2(n)$ time.*

[a] If $t(n) \leq n$ then we don't have time to read the input - so this is a reasonable assumption.

Proof. No Proof. □

Theorem 5.2. *If $t(n) \geq n$, and a nondeterministic TM that runs within $t(n)$ time, there is a deterministic single strip TM that runs within $2^{O(t(n))}$ time.*

Definition 5.2 (Nondeterministic Complexity classes)**.** Define:

- $\text{NTIME}(t(n)) = \{\mathcal{L} \mid \exists \mathcal{M} \text{ nondeterministic decider that for } |w| = n| \text{ halts after } O(n) \text{ steps}\}$
- $\text{PTIME} = \bigcup_{k \geq 0} \text{TIME}(n^k)$, with the petname P.
- $\text{NPTIME} = \bigcup_{k \geq 0} \text{NTIME}(n^k)$, with the petname NP.

[I] In every iteration - delete half of the 0s and 1s that survived until the current stage
[II] In fact - $\mathcal{L} \in \text{TIME}(g(n))$ for $g \in o(n \log n)$ then $\mathcal{L} \in \text{REG}$.

- EXPTIME $= \bigcup_{k \geq 0} \text{NTIME}(2^{n^k})$

Theorem 5.3. *$P \subseteq NP \subseteq EXPTIME$*

Proof. Any deterministic TM is a nondeterministic TM (first containment) and any nondeterministic TM can be simulated by a deterministic TM in exponential time (second containment). □

5.2 The classes P, NP

Consider two classical problems on graphs: Euler paths and Hamiltonian paths. Let $G = (V, E)$ (simple undirected graph). Let $\mathcal{G}_E, \mathcal{G}_H$ be the languages of graphs with Euler and Hamilton paths (respectively), then $\mathcal{G}_E \in$ P (even NTIME($O(n)$)). But we don't know $\mathcal{G}_H \overset{?}{\in}$ P. With that said - we do know $\mathcal{G}_H \in$ NP by guessing a permutation of the vertices.

Example 5.2. Define the language

$$D - ST - HAMPATH = \{\langle G, s, t\rangle \mid G \text{ is a Directed graph with hamiltonian path } s \to t\}$$

For example - this digraph has an Hamiltonian $s \to t$ path:

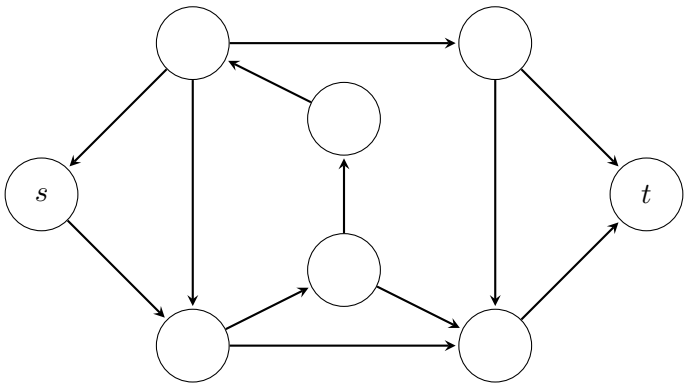


Figure 5.1: Graph with Hampath $s \to t$

Proposition. *$D - ST - HAMPATH \in EXPTIME$*

Proof. Given G, s, t, let $\pi_1 \dots \pi_{n!}$ denote the permutations of $V(G)$:

1. While $i \leq n!$:
2. Let $\pi_i = v_1^i \dots v_n^i$
3. If $v_1^i = s, v_n^i = t$ and $(v_j^i, v_{j+1}^i) \in E$ - accept. Otherwise $i+ = 1$.

This algorithm runs in exponential time. □

Note that it is easy to check a given permutation. The reason that the algorithm is exponential is due to the number of possibilities.
Also - $D - ST - HAMPATH \in$ NP. Consider the NTM that guesses a permutation π and checks if it is an s, t Hamiltonian path.
What about $D - ST - HAMPATH^c$ (all graphs with no Hamiltonian path). Once again it is in EXPTIME, but[III] not in NP!

[III] Probably

Example 5.3. $COMPOSITE = \{x \mid x = p \cdot q\}$, of course $COMPOSITE^c = PRIME$. By assumption - x is given in binary, so $COMPOSITE \in$ NP: Deciding prime numbers for an unary represented number is indeed polynomial. Since the input's length (in binary) is logarithmic in the unary representation - the time complexity is actually exponential in the representation.

Definition 5.3 (Verifier)**.** For a language $\mathcal{L}$, a TM $\mathcal{V}$ is a **Verifier** of $\mathcal{L}$ if

$$\mathcal{L} = \{w \mid \mathcal{V} \text{ accepts } \langle c, v\rangle \text{ for some } x \in \Sigma^*\}$$

The verifier's complexity is measured WRT $|w|$: That is, runs in $poly(|w|)$ time on $\langle w, c\rangle$. This implies $|c| = poly(|w|)$.

Theorem 5.4 (Alternative characterization)**.** $\mathcal{L} \in NP \iff$ *there exists a polynomial verifier for* $\mathcal{L}$.

Proof. Assume $\mathcal{V}$ is a verifier. Construct a polynomial NTM that guesses a word c with $|c| = poly(|w|)$ (all words with $|c| \leq f(|w|)$ where f is $\mathcal{V}$'s time complexity) and runs $\mathcal{V}$ over $\langle w, c\rangle$. For the other side - let $\mathcal{L} \in$ NP, then define $\mathcal{V}$ to accept $\langle w, c\rangle \iff c$ is an accepting run of $\mathcal{M}_{\mathcal{L}}$ on w. Correctness is obvious, and since $\mathcal{M}_{\mathcal{L}}$ is polynomial - $|c| = poly(|w|)$, hence $\mathcal{V}$ is polynomial. □

Example 5.4. A verifier for $D - ST - HAMPATH$ is the machine $\mathcal{V}$ with

$$\mathcal{L}(\mathcal{V}) = \{\langle G, s, t, \pi\rangle \mid \pi \text{ is a Hamiltonian } s,t \text{ path}\}$$

Example 5.5. A verifier for $COMPOSITE$:

$$\mathcal{L}(\mathcal{V}) = \{\langle x, p, q\rangle \mid x = p \cdot q \quad q, p \neq 1\}$$

Of course there are variations and different verifiers (e.g - only one devisor)

5.3 NP-Completeness

We would like to show that some things are not in P. However - this is hard: We would need to prove that no polynomial TM decides the language. For this - we define a class of languages that had they been in P, then things break beyond repair. For that - we need some more definitions. Until now (in computability theore) we used reductions to show that some problems are not in R, or not in RE. Now we care about time complexity - so we need to take care of that.

Definition 5.4 (Polynomialy Computable)**.** A function $f : \Sigma^* \to \Sigma^*$ is said to be **Polynomialy computable** if there is a polynomial TM $\mathcal{M}$ that computes it.

Definition 5.5 (Polynomialy Reducable)**.** We say A is **Polynomialy Reducable** to B (denote $A \leq_p B$) if there is a poly-computable function $f : \Sigma^* \to \Sigma^*$ with

$$w \in A \iff f(w) \in B$$

Theorem 5.5 (Reduction Theorem)**.** *If $A \leq_p B$ and $B \in P$ then $A \in P$.*

Proof. Let $\mathcal{M}_b$ poly TM for B with complexity function t_B, and f a poly-reduction $A \leq_p B$. A TM that decides A: Given w compute $f(w)$ in $t_f(|w|)$ time, and $f(|w|) \leq t_f(w)$. Now run $\mathcal{M}_B$ over $f(w)$ in no more than $t_b(t_f(w))$ which is polynomial in $|w|$. □

With these definitions, we can define "hard languages":

Definition 5.6 (NP - hard)**.** We say that $\mathcal{L}$ is **NP-Hard** if

$$\forall \mathcal{L}' \in \text{NP} \quad \mathcal{L}' \leq_p \mathcal{L}$$

Definition 5.7 (NP - complete)**.** We say that $\mathcal{L}$ is **NP-Complete** if $\mathcal{L} \in$ NP and is NP-hard.

Theorem 5.6. *Let $\mathcal{L} \in NP$. Then $\mathcal{L} \in$ NP-complete $\iff$ there exists an NP hard language $\mathcal{L}'$ such that $\mathcal{L}' \leq_p \mathcal{L}$.*

Proof. $\Leftarrow$ By transitivity. $\Rightarrow$ By reflexivity. □

Example 5.6. define $SAT = \{\langle \Theta \rangle \mid \Theta \text{ is satisfyable}\} \in$ NP (by guessing an assignment). This language is NP complete - we will show this later.

Definition 5.8 (Conjunctive Normal Form)**.** A **literal** is a Boolean variable or a negated Boolean variable, as in x or $\neg x$. A **Clause** is several literals connected with $\vee$s, as in $(x_1 \vee x_2 \vee x_3 \vee \neg x_4)$. A boolean formula is in **conjunctive normal form**, called a **cnf-formula**, if it comprises several clauses connected with $\wedge$s

Example 5.7. Let $3SAT = \{\langle \varphi \rangle \mid \varphi \in 3CNF\}$[IV]
Define $CLIQUE = \{\langle G, k \rangle \mid$ The graph G has a clique of size $k\}$. We show $3SAT \leq_p CLIQUE$.
It is easy to show that $3SAT \leq_m CLIQUE$: Given Θ, check if it is satisfyable (in exponential time). If so - write $\langle K_2, 2 \rangle$ on the strip. If not - write $\langle K_2, 3 \rangle$.
We now show a polynomial reduction $3SAT \leq_p CLIQUE$. Let $f : \varphi \mapsto \langle G, k \rangle$ be the following function: Given

$$\varphi = \bigwedge_{i \in [m]} (\ell_i^1 \vee \ell_i^2 \vee \ell_i^3)$$

Define G with $V(G) = \{\ell_i^1, \ell_i^2, \ell_i^3 \mid i \in [m]\}$ (all the literals in φ), let $k = m$ (the number of clauses) and

$$E(G) = V \times V \setminus (\{(u, v) \mid u, v \text{ are in the same clause}\} \cup \{(u, v) \mid u = \neg v\})$$

This build is polynomial- since there are $3m$ vertices and $O(m^2)$ edges. As for correctness, assume φ is satisfyable and let $\nu : X \to \{T, F\}$ a satisfying valuation of φ. Thus, in any clause C_j of φ there is at least one literal $\ell_j^{k_j}$ with $\nu(\ell_j^{k_j}) = T$ (since it is in CNF).

[IV] That is, φ is in CNF form with 3 literals in any clause.

Proposition. *The collection* $\left\{\ell_j^{k_j}\right\}_{j\in[m]}$ *defines a clique in* $G(\varphi)$

Proof. Indeed, $(\ell_j^{k_j}, \ell_i^{k_i}) \in E$ since they are not from the same clause and cannot be negations of one another - since they are valued the same under ν. □

Conversely, assume G has a clique of size m. By definition of E, the clique contains exactly one representative of any clause C_j, and the representatives are consistent under valuations. This defines a satisfying valuation ν of φ (exercise).

An example of this build is shown:

Figure 5.2: The build from $\varphi = (x_1 \vee x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_2)$

Theorem 5.7. *$\mathcal{L}$ is NP-complete $\Rightarrow$ ($\mathcal{L} \in P \Rightarrow P = NP$).*

Proof. Let $\mathcal{L}$ an NP-complete language. Hence, for any $\mathcal{L}' \in$ NP there is a poly-reduction $f_{\mathcal{L}'}$ from $\mathcal{L}'$ to $\mathcal{L}$. Therefore $\mathcal{L} \in$ P implies that $\mathcal{L}' \in$ P. Since P $\subset$ NP we get P = NP. □

Example 5.8. Consider the language of vertex cover $VC = \{\langle G, k\rangle \mid G$ has a vertex cover of size at most $k\}$. We show that this language is NP-Complete.

Proposition. *$VC \in NP$*

Proof. Consider the verifier that gets as input $\langle G, k, S\rangle$ with $S \subset V$ and checks that $|S| \leq k$, then traverses E and checks for $e \in E$ if there is a vertex $v \in V$ that is incident with e. If this holds for all $e \in E$, accept, otherwise reject. This is clearly correct. As for time complexity - checking the size of S is at most $|V(G)|$. For any edge we compare it against all V, so total runtime of $O(|E| \cdot |V|)$, hence the runtime is $poly(|\langle G\rangle|)$ □

Proposition. *VC is NP-hard.*

Proof. We show that $CLIQUE \leq_p VC$, which results in VC being NP hard. Consider $f : \langle G, k\rangle \mapsto \langle G^c, n-k\rangle$ with G^c the complement graph of G.
Assume G has a clique of size k and $v_1 \dots v_k$ clique vertices in G. No two of them are connected in G^c, then taking $S = V \setminus \{v_i\}_{i\in[k]}$ is hence a vertex cover of G^c, of size $n-k$. Conversely - if G^c has a vertex cover of size $n-k$, denoted $S = \{v_1 \dots v_{n-k}\}$, then all $V \setminus S$ define a clique: If $x, y \in V \setminus S$, we want to show $\{x, y\} \in E$, that is $\{x, y\} \notin E(G^c) = E^c$. If $\{x, y\} \in E^c$, since $x, y \notin S$ - S is not a vertex cover of G^c, contradiction. Hence $V \setminus S$ defines a k-clique. As for time complexity - inverting the edges takes $O(|E|)$ time, and $k \mapsto n-k$ takes $O(|V|)$. □

Definition 5.9. Given G, a dominating set $D \subset V$ is a set such that for all $v \in V$, $d(v, S) \leq 1$.

Example 5.9. Consider $\{\langle G, k\rangle \mid$ There exists a dominating set in G of size $k\}$. We show that DS is NP complete.
A polynomial verifies would receive $\langle G, k, C\rangle$ and checks that C is a dominating set: Over $v \in V$ check if $v \in C$, if not - check if for any edge incident with v if the other incident is in C. If

exists - accept. Otherwise, reject. This is $poly(|G|)$: Iterating over V is $O(|V|)$, over the edges is $O(|C|) = O(|V|)$, then total runtime of $O(|V|^2|E|)$.
We reduce $VC \leq_p DS$. Consider the mapping $f : \langle G, k\rangle \mapsto \langle G', k'\rangle$: for $e = (x, y) \in E(G)$, f adds a new vertex v_e and the edges $\{(x, v_e), (y, v_e)\}$ (we only enrich the graph). Then f returns G' with $V(G') = V \cup \{v_e\}_{e\in E(G)}$ and $E(G') = E \cup \{(x, v_e), (y, v_e)\}_{e\in E(G)}$. Finally - f counts the isolated nodes in G (say, m) and returns $\langle G', k + m\rangle$.
Time complexity is $O(|V| + 3|E|) = O(V + |E|) = poly(|G|)$.
As for correctness - assume G has a VC C of size k, and denote F the isolated nodes of G: $C \cup F$ is a DS in G' of size at most $k + m$: For $v \in V(G')$, it is either in F (isolated), or not. If it is not, and $v \in V(G)$ - v is in an edge $\{v, u\}$ and since C is a VC of G, and v is of distance 1 to C. If $v = v_e$ for some e, then $e = \{x, y\}$, WLOG $x \in C$ and thus the distance to C is at most 1.

Conversely - assume G' has a dominating set of size k' - it must contain the isolated vertices of G, discard them and we are left with a set S of size k. If some $v_e \in S$, substitute it for some $x \in e = \{x, y\}$. This is a dominating set (proof omitted for I am awfully tired).

5.4 Cook - Levin Theorem

Theorem 5.8 (Cook - Levin Theorem)**.** $3SAT \in$ *NP-complete.*

Proof. First, $3SAT \in$ NP by the verifier $\mathcal{V}$ with the language $\langle\varphi, \nu\rangle$ where ν is a satisfying valuation of φ. It is left to show $3SAT \in$ NP-hard.
Let $\mathcal{L} \in$ NP. Define $f : w \mapsto \varphi \in 3CNF$. φ is satisfyable if there exists a valuation, and $w \in \mathcal{L}$ if there exists an accepting run over $\mathcal{M}$ NTM that decides $\mathcal{L}$ within $t(|w|) = t(n)poly(n)$ time..
So we need to match runs to valuations of φ. That is - there exists a sequence of configurations $C_0 \dots C_m$ which is an accepting run of $\mathcal{M}$ over w, and $m = t(n)$. We build a matrix that in its entries there are characters from $S = \{\#\} \cup Q \cup \Gamma$:

Figure 5.3: The tableux of configurations. We indexed from lower to upper, this is from the book.

The dimensions of this matrix is $t(n) \times (t(n) + 3)$ (at most). The formula φ will say "there exists such matrix of an accepting run". Let us construct it:
Its variables are:

$$\forall i, j \text{ entry in the matrix, and } s \in S \text{ - assign a variable } x_{i,j,s} \text{ that describes } A_i^j = s$$

$\varphi = \varphi_{cell} \wedge \varphi_{init} \wedge \varphi_{acc} \wedge \varphi$ With:

- φ_{cell} says "a satisfying valuation describes an accepting run" (that is, no $\nu(x_{1,1,a}) = \nu(x_{1,1,b})$)
- φ_{init} says "an assignment to the first row complies with the initial configuration of $\mathcal{M}$ over w".
- φ_{acc} says "there is a row that describes an accepting configuration".
- φ_{move} says "ascending in the matrix complies with transition between subsequent configurations".

So:

$$\varphi_{cell} = \bigwedge_{i,j} \left[\overbrace{\left(\bigvee_{s \in S} x_{i,j,s} \right)}^{\text{Every cell contains a letter}} \wedge \overbrace{\left(\bigwedge_{s_1 \neq s_2} \neg x_{i,j,s_1} \vee \neg x_{i,j,s_2} \right)}^{\text{Only one letter}} \right]$$

$$\varphi_{init} = \overbrace{x_{1,1,\#} \wedge x_{2,1,q_0} \wedge x_{3,1,w_1} \wedge \ldots x_{n+2,n,w_n} \wedge x_{n+3,1,\sqcup}, \ldots \wedge x_{t(n)+3,1,\#}}^{\text{The first row is the initial configuratoin}}$$

$$\varphi_{acc} = \overbrace{\bigvee_{j \in [t(n)]} \left[\bigvee_{2 \leq i \leq t(n)+2} x_{i,j,q_{acc}} \right]}^{\text{In some row there is } q_{acc}}$$

For φ_{move} - it is sufficient to check submatrices of size 3×2 (since legal transitions are very local). That is - there is a set W of finite cardinality of legal "windows" (submatrices) of size 3×2. φ_{move} will say "all windows of size 3×2 are from W":

$$\varphi_{move} = \bigwedge_{\substack{j \leq t(n)-1 \\ i \leq t(n)+1}} \left[\bigvee_{(s_1,s_2,s_3,s_4,s_5,s_6) \in W} \overbrace{x_{i,j,s_1} \wedge x_{i+1,j,s_2} \wedge x_{i+2,j,s_2} \wedge x_{i,j+1,s_4} \wedge x_{i+1,j+1,s_5} \wedge x_{i+2,j+1,s_6}}^{\text{the window is legal}} \right]$$

We now define W (and that it is of polynomial size in n). For any transition $\delta(q_1, a) \ni \langle q_2, b, R \rangle$ (and of course similarly for L), add windows of the following types:

Figure 5.4: Examples of legal windows

And in sequential notation (for example) - (q_1, a, c, b, q_2, b). So for any transition there is a finite number of valid windows. In fact - this is independent of $|w|$ - so we're all good.
We show that φ is satisfyable iff there is an accepting run of $\mathcal{M}$ over w: Assume f is a truthful valuation of φ, then f describes a matrix of size $(t(n) + 3) \times t(n)$. Since φ_{init} is satisfied - the first row describes the initial configuration of $\mathcal{M}$ on w. Since φ_{move} is satisfied - the $j + 1$'s row is a subsequent configuration of the j's row. Since φ_{acc} is satisfied - at some point, we reach an accepting configuration. Therefore - there is an accepting run of $\mathcal{M}$ on w.
Conversely - if there exists an accepting run of $\mathcal{M}$ over w, there is a matrix with the sequence of configuration describing a satisfying valuation of φ.

All constructions are polynomial in $t(n)$, hence polynomial in n - so the reduction is polynomial.

φ is in CNF form, but not $3CNF$. We will show in the exercise that we can reform a formula in CNF to a formula in $3CNF$ in polynomial time. □

5.5 More examples

As we defined NP hardness and NP completeness, there are symmetric definitions for coNP (as well as for any complexity class). We see some examples:

Example 5.10. Define $VAL = \{\langle\varphi\rangle \mid \text{All valuations satisfy } \varphi\}$. That is, for any $\nu : X \to \{T, F\}$ we have $\nu \vDash \varphi$, iff there is no ν with $\nu \vDash \neg\varphi$ iff $\varphi \notin SAT$. So $VAL \in$ coNP

Theorem 5.9. *$\mathcal{L}$ is coNP-complete $\iff$ $\mathcal{L}^c$ is NP-complete.*

Proof. By definition, for any $\mathcal{L}' \in$ coNP we have $\mathcal{L}'^c \in$ NP. Therefore there is a reduction $\mathcal{L}'^c \leq_p \mathcal{L}^c$ since $\mathcal{L}^c$ is NP-hard. The same reduction implies $\mathcal{L}' \leq_p \mathcal{L}$, hence $\mathcal{L}$ is coNP hard. □

Another reduction $3SAT \leq_p CLIQUE$

Given $\varphi = \bigwedge_{i \leq j \leq m} \left(\ell_j^1 \vee \ell_j^2 \vee \ell_j^3\right)$, we would like to define an undirected graph G and a number k such that $K_k \subset G$ iff φ is satisfyable. Consider the clause $C_j = x_1 \vee \neg x_2 \vee x_3$. There are $2^3 - 1$ satisfying valuations[V]. We will consider these valuations as **restrictions**. Denote $F_j = \{\nu_j^l\}_{l \in [7]}$ the satisfying valuations of C_j (over its three variables). Define $V(G) = \bigcup_{j \in [m]} F_j$.

I can't find the example in my photos - If anyone could send it to me, that'd be great

And let $E(G)$ be the set of edges with no "contradiction", that is $\{\nu_j^{l_1}, \nu_t^{l_2}\} \in E(G)$ agree on their common variables (and of course no loops). We claim $K_m \subset G$ iff there is a satisfying valuation.

Note that this construction is polynomial in $|\varphi|$ - since there are at most $8 \cdot m$ valuations to check (to defnie nodes), and the edges need at most $O(3 \cdot m^2)$ things to check.

Remark. This construction is exponential in 3 - that is, for a fixed k, this will work for any $k - CNF$ class of formulas.

[V] Consider the equivalent disjunctive form of the clause

Chapter 6

Space Complexity

6.1 Introduction

Definition 6.1 (Space Complexity)**.** We say that a (single strip) TM $\mathcal{M}$ works in **Space Complexity** $S : \mathbb{N} \to \mathbb{N}$ if for any input w, $\mathcal{M}$ uses at most $S(|w|)$ cells of the strip.

As in time complexity - there are space complexity classes.

Definition 6.2. Given $S : \mathbb{N} \to \mathbb{N}$, we define

$$\text{SPACE}(S(n)) = \langle \mathcal{L} \mid \mathcal{L} \text{ is decided by a TM with space compexity } O(S(n)) \rangle$$

Theorem 6.1. $TIME(f(n)) \subset SPACE(f(n))$

Proof. If a TM works in time $f(n)$, it can reach with its head only to $f(n)$ cells. □

Theorem 6.2. $SPACE(f(n)) \subset TIME(2^{O(f(n))})$

Proof. We compute how many configurations exists to a TM $\mathcal{M}$ with space complexity $S(n)$: $|Q| \cdot S(n) \cdot |\Gamma|^{S(n)} = c_1 \cdot S(n) \cdot c_2^{S(n)} = 2^{O(S(n))}$.
Since $\mathcal{M}$ decides a language and is deterministic, it does not repeat the same configuration twice. Therefore - it halts within $2^{O(S(n))}$ steps. □

6.2 Space Complexity Classes

Definition 6.3. We define, similarly to time complexity classes:

- $\text{NSPACE}(S(n)) = \{\mathcal{L} \mid \mathcal{L} \text{ i decided by a nondeterministic TM with } O(S(n) \text{ space})\}$
- $\mathsf{PSPACE} = \bigcup_{k \in \mathbb{N}} \text{SPACE}(n^k)$

- $\mathsf{NPSPACE} = \bigcup_{k\in\mathbb{N}} \mathrm{NSPACE}(n^k)$

Remark. By the previous theorems, we know:

$$PTIMES \subset \mathsf{PSPACE} \qquad \mathsf{PSPACE} \subset EXPTIME$$

Example 6.1. $SAT \in \mathsf{PSPACE}$. The idea is to check all valuations of a given formula (by using the same space) one after the other (time-wise). Let $\nu_1 \dots \nu_{2^n}$ an ordering of the valuations (of φ with n variables), such that given ν_i, we can find ν_{i+1} with linear space - or return $\perp$ if we are in the last valuations (eg - lexicographic order). $\mathcal{M}$ operates the following way: One strip will hold ν_1, the other the value $\nu(\varphi)$. While the first strip does not say $\perp$, $\mathcal{M}$ computes $\nu_i(\varphi)$. If T -halt and accept. Otherwise - update in the first strip to the valuation ν_{i+1}. When the first strip contains $\perp$ - reject.

Note that the first strip requires $O(n)$ cells (and the space required for an update). The second strip needs $O(|\varphi|)$ cells for computing $\nu(\varphi)$.

This can easily be extended for any language in NP.

Theorem 6.3. *$NP \subset PSPACE$*

Proof. Let $\mathcal{L} \in$ NP, ad let $\mathcal{V}$ be a polynomial verifier of $\mathcal{L}$, with $f : \mathbb{N} \to \mathbb{N}$ bounding $\mathcal{V}$'s runtime. Let $c_1 \dots c_{|\Sigma|^{f(|w|)}}$ the lexicographic ordering of $\Sigma^{\leq f(|w|)}$. A TM that decides $\mathcal{L}$ in polyspace would go over all c_i (the "witnesses") and simulate $\mathcal{V}$ to check them. The details can be completed as in the previous example. □

Example 6.2. Denote $EMPTY_{NFA} = \{\langle\mathcal{A}\rangle \mid \mathcal{L}(\mathcal{A}) = \emptyset\}$ and $ALL_{NFA} = \{\langle\mathcal{A}\rangle \mid \mathcal{L}(\mathcal{A}) = \Sigma^*\}$ and consider their complements $\overline{EMPTY_{NFA}}, \overline{ALL_{NFA}}$. It is clear that $\overline{EMPTY_{NFA}} \in$ NP - by the witness in the form of a word (of length corresponding to a simple path over $\mathcal{A}$'s graph, therefore polynomial in $\langle\mathcal{A}\rangle$). We need to show that it can be verified in polytime (since $\mathcal{A}$ is nondeterministic it is not trivial). Consider the automaton $\mathcal{A}_w$ with $\mathcal{L}(\mathcal{A}_w) = \{w\}$. Then the product automaton $\mathcal{A} \times \mathcal{A}_w$[I] has the language $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{A}_w)$. So $w \in \mathcal{L}(\mathcal{A})$ iff $\mathcal{L}(\mathcal{A} \times \mathcal{A}_w) \neq \emptyset$. Checking this is in fact checking if an accepting state is reachable from some initial state. In Algorithms course we saw that this is polynomial in the graph's size.

For $\overline{ALL_{NFA}}$, how would we go about that? For $\overline{ALL_{DFA}}$ this is easy (since the complement automaton would have empty language). It is not clear that $\overline{ALL_{NFA}} \in$ NP. We will construct NFA $\mathcal{A}$ with $\mathcal{L}(\mathcal{A}) \neq \Sigma^*$, but the shortest word $\mathcal{A}$ rejects is exponential in $|\mathcal{A}|$:
Let $i \geq 1$, consider the first i prime numbers $p_1 \dots p_i$, and consider the language:

$$\mathcal{L}_i = \{w \mid |w| \text{ is not divisible by one of } p_1 \dots p_i\}$$

Denote $\mathcal{A}_{p_j}$ the automaton with a cycle of size p_j and the first state rejects. Then $\mathcal{A}_i = \bigsqcup_{j\in[i]} A_{p_j}$, and since $|p_j| = O(j \log j)$, $|\mathcal{A}_i|$ is polynomial in i. Consider $w^* = a^{\prod_{j\in i} p_j}$, then $w^* \notin \mathcal{L}_i$, but $|w^*| \geq 2^i = \exp(i)$. Then there is a word which is not in $\mathcal{L}_i = \mathcal{L}(\mathcal{A}_i)$. We need to show that a^y with $y < \prod_{j\in i} p_j$ is accepted. This is because when we decompose y into prime components, there must appear a component which is not in $\{p_j\}_{j\in[i]}$.

Claim 6.2.1. *$\overline{ALL_{NFA}} \in NPSPACE$.*

[I] A variation of this

Proof. Given $\mathcal{A}$ NFA, the following are equivalent:

1. $\mathcal{L}(\mathcal{A}) \neq \Sigma^*$
2. $\mathcal{L}(\overline{\mathcal{D}}) \neq \emptyset$[II] with $\mathcal{D}$ the DFA generated by the subset construction over $\mathcal{A}$.
3. There exists an increasing sequence of states $\{Q_i\}_{i=0}^{k}$ with $k \leq 2^{|Q|}$ such that Q_0 is the set of initial states of $\mathcal{A}$, and for all i, $Q_{i+1} = \delta(Q_i, \sigma_i)$ for some σ, and $Q_k \cap F = \emptyset$ (that is, this is a sequence corresponding to an accepting run of some word in $\overline{\mathcal{D}}$).

(1 $\iff$ 2) Since $\mathcal{L}(\overline{\mathcal{D}}) = \Sigma^{\setminus}\mathcal{L}(\mathcal{A})$, this is immediate.
(2 $\iff$ 3) $\mathcal{L}(\overline{\mathcal{D}}) \neq \emptyset$ $\iff$ there is a word with length $\leq 2^{|Q|}$ (number of states of $\overline{\mathcal{D}}$) that is accepted (because there is a word that's accepted over a simple path) $\iff$ there exists the required sequence.

Then define an NTM operating in polyspace $\mathcal{M}$ and decides $\overline{ALL_{NFA}}$: On the strip we will hold a counter (until $2^{|Q|}$, requires $O(|Q|)$ space!) and the encoding of the current state $S = Q_i$ (an instance of a subset of Q, lets say represented by a subset of $[n]$ - still polyspace in $|Q|$)

1. Write Q_0 on the strip (where we keep states)
2. writes $i = 0$ (where we keep counter)
3. While $i \leq 2^{|Q|}$:
   3.1 If $Q_i \cap F = \emptyset$ - halt and accept
   3.2 Otherwise - **guess** a letter $\sigma \in \Sigma$ and updates the strips: Q_i is substituted with $\delta(Q_i, \sigma)$ and $i++$.
4. Halt and reject

□

Theorem 6.4 (Savitch Theorem)**.** *For any space complexity S with $S(n) \geq n$,*

$$NSPACE(S(n)) \subset SPACE(S^2(n))$$

Proof. Let $\mathcal{N} \in \text{NSPACE}(S(n))$. Preliminaries:

- For any w, $\mathcal{N}$ has a unique initial configuration C_{init}^w.
- $\mathcal{N}$ has a unique accepting configuration (say, when $\mathcal{M}$ arrives at an accepting configuration - it cleans the strip and goes back to the beginning of the strip), denoted C_{acc}.
- Let d be such that $\mathcal{N}$ has at most $2^{d \cdot S(n)}$ different configurations during a run over a word of length n.

We build a deterministic procedure[III] $\mathcal{M} = reach(C_1, C_2, t)$ that decides if the configuration C_2 is reachable from C_1 within t steps. Its space complexity would be $O(\log(t) + S(n)) \cdot \log(t)$. Then given w, an NTM $\mathcal{M}'$ would check $reach(C_{init}^w, C_{acc}, 2^{d \cdot S(|w|)})$. Note that indeed $\mathcal{M}$ accepts w iff $reach(C_{init}^w, C_{acc}, 2^{d \cdot S(|w|)})$, and that the space complexity would be $O(S^2(n))$.

The procedure:

[II] The accepting states of $\overline{\mathcal{D}}$ are exactly $S \subset Q$ such that $S \cap F = \emptyset$
[III] A part of a TM, we will iteratively run this procedure

1. If $C_1 = C_2$ or C_2 is subsequent of C_1 - accept.
2. Otherwise - Iterate over all configurations C that use $S(|w|)$ cells. For each of them:
   2.1 Check if $reach(C_1, C, \lceil \frac{t}{2} \rceil)$
   2.2 Check if $reach(C, C_2, \lfloor \frac{t}{2} \rfloor)$
   2.3 If both accepted - accept.
3. Reject

Correctness is obvious. The hard part is checking the space complexity. The recursion's depth is $log(t)$. The recursion "saves" only one branch of the recursion tree in any given moment. Hence we save only the path from the root to the current node, and information related to iterating over all configurations. That is - to save a single node, we need:

$$2 \cdot S(n) + \log(t)$$

space to save any node (two configurations $S(n)$ and current path length $\log(t)$), information to iterate over configurations is $S(n)$, so total space complexity $O(S(n) + \log(t)) \cdot \log(t)$.
How does *reach* "knows" $S(n)$? If $S(n)$ is known - everything's fine. If $S(n)$ is unknown, build the same TM with a twist. Note that *reach* can be used to check if there exists a reachable configuration that uses i cells: Iterate over all configurations that uses i cells, and for each such C, check $reach(C^w_{init}, C, 2^{di})$ (assuming $S(n) = i$). We build an outer loop to deterministically decide if $\mathcal{M}$ accepts w:

1. For $i \in \mathbb{N}$:
   (a) Check if there is a reachable configuration that uses i cells? If not - reject
   (b) If there is - run the previously built $\mathcal{M}$ with $S(n) = i$. If accepted - accept. Otherwise, $i++$.

□

Corollary 6.5.

$$NPSPACE = PSPACE = coNPSPACE = coPSPACE$$

6.3 Space-Hardness and Space-Completeness

Definition 6.4 (PSPACE-hard)**.** We say that $\mathcal{L}$ is **PSPACE-hard** if for any $\mathcal{L}' \in$ PSPACE, $\mathcal{L}' \leq_p \mathcal{L}$.

Remark. The same notation of $\leq_p$ as in time complexity.

Definition 6.5 (PSPACE - complete)**.** We say that $\mathcal{L}$ is **PSPACE-complete** if $\mathcal{L} \in$ PSPACE and is PSPACE-hard.

We will see a PSPACE-complete language. For this we need a new definition regarding boolean formulas:

Definition 6.6. We say a formula φ is **Fully Quantified** if any variable $x \in \varphi$ is connected to a quantifier $\exists, \forall$

Example 6.3. Define $TQBF = \{\langle\varphi\rangle \mid \varphi \text{ is a True fully quantifid formula}\}$. That is, $\varphi \in TQBF$ is a totally quantified tautology.

Claim 6.3.1. *$TQBF \in PSPACE$*

Proof. Define $\mathcal{M}$ to operate recursively over φ:

1. (Base Case) If φ is without quantifier, then $\mathcal{M}$ valuates φ and accepts if True. Otherwise - reject.
2. If $\varphi = (\exists x)\psi$, then run $\mathcal{M}(\psi)$ recursively twice: once with $x = 0$ and once with $x = 1$. Accept if one of them accepts - accept. Otherwise reject.
3. If $\varphi = \forall x\psi$ then run $\mathcal{M}(\psi)$ recursively with $x = 0$ and with $x = 1$. Accept if both accepted. Otherwise reject.

This obviously decides $TQBF$. Note that the depth of enery branch in the recursion tree is bounded by number of variables of $\varphi \leq |\varphi|$, and for any variable we run at most twice. In every stage we remember which variables were substituted and what's left to check - $O(|\varphi|)$ space. Finally, in the base case we valuate φ which takes $O(|\varphi|)$ space - all in all, the space complexity is $O(|\varphi|^2)$ - as required. □

We would like not wo show it is complete in PSPACE. How do we encode configurations using boolean formulas?
Let $\mathcal{M}$ be a TM and assume $\mathcal{M}$ uses S strip cells. Encode the following way:

1. For $i \in [S]$ define a variable x_{ia} which encodes "a" is written in cell i".
2. For $i \in [S]$ define a variable y_i which encodes "The reading head is over cell i"
3. For any $q \in Q$ define a variable z_q which encodes "The machine $\mathcal{M}$ is in state q".

Theorem 6.6. *Let $\mathcal{M}$ be a TM. Then:*

1. *There exists a formula $\varphi_{valid}(C)$ which is True iff C is a valid encoding of a configuration of $\mathcal{M}$.*
2. *There exists a formula $\varphi(C_1, C_2)$ which is True iff C_2 is subsequent of C_1.*

In addition - both formulas are computable in polyspace in S.

Proof. Define:

$$\varphi_{valid}(C) = \bigwedge_{i\in[S]} \bigvee_{a\in\Gamma} \left(x_{ia} \wedge \bigwedge_{b\in\Gamma\setminus\{a\}} \neg x_{ib} \right) \wedge \bigvee_{i\in[S]} \left(y_i \wedge \bigwedge_{j\in[S]\setminus\{i\}} \neg y_j \right) \wedge \bigvee_{q\in Q} \left(z_q \wedge \bigwedge_{q'\in Q\setminus\{q\}} z_{q'} \right)$$

Note that $|\varphi_{valid}(C) = O(S^2)$. Now define for any $i \in [S], a \in \Gamma, q \in Q$, if $\delta(q, a) = (r, b, R)$ then define:

$$\psi_{iaq}(C_1, C_2) = (x^1_{ia} \wedge y^1_i \wedge z^1_q) \Rightarrow (x^2_{i,b} \wedge y^2_{i+1} \wedge z_r \bigwedge_{j\in[S]\setminus\{i\}} \bigwedge_{d\in\Gamma} (x^1_{jd} \iff x^2_{jd}))$$

And of course, symmetrically define for transition to L, and for exceptions (e.g - when we are in cell 1 and move to the left). Convince yourself that this actually encodes the transition. Now define:

$$\varphi(C_1, C_2) = \varphi_{valid}(C_1) \wedge \varphi_{valid}(C_2) \wedge \bigwedge_{i\in[S]} \bigwedge_{a\in\Gamma} \bigwedge_{q\in Q} \psi_{iaq}(C_1, C_2)$$

Once again - the length of $\psi(C_1, C_2) = O(S^2)$ □

Claim 6.3.2. *$TQBF$ is PSPACE-hard.*

Proof. Let $\mathcal{L} \in$ PSPACE be a language. We would like to build a function $f : w \mapsto \varphi_w$ such that $w \in \mathcal{L} \iff \varphi_w \in TQBF$. Let $\mathcal{M}$ be a TM that decides $\mathcal{L}$ is polyspace.
Define a formula that says "C_2 is reachable from C_1 within k steps":

$$\varphi_k(C_1, C_2) = \exists C_m \forall C_3, C_4 \left[(C_3 = C_1) \wedge (C_4 = C_m) \vee (C_3 = C_m) \wedge (C_4 = C_2)\right] \Rightarrow \varphi_{k/2}(C_3, C_4)$$

Any recursive call is polynomial and there is only one branch - so no exponential number of leaves. This formula is built in polyspace in w. □

Example 6.4. Consider the language $\overline{ALL_{NFA}}$. We've shown that $\overline{ALL_{NFA}} \in$ NPSPACE. By Savitch - $\overline{ALL_{NFA}}, ALL_{NFA} \in$ PSPACE. We will see that $\overline{ALL_{NFA}}$ is PSPACE-hard.

6.4 Sublinear Space Complexity

Definition 6.7. Define

$$LOGSPACE = \text{SPACE}(\log(n)), NLOGSPACE = \text{NSPACE}(\log(n))$$

denoted L, NL respectively.

Remark. Since the input itself takes $|w|$ space, we change the computational model a little: We require two strips, and that the work strip would be of logarithmic space.

Example 6.5. Let $EQ = \{0^k 1^k \mid k \in NN_0\}$. We saw that $EQ \in$ P, but the algorithm wrote on all of the cells, thus $EQ \in \text{SPACE}(n)$. We define a TM $\mathcal{M}$ that decides EQ in $LOGSPACE$. The machine maintains two counters C_0, C_1 in binary, and counts. each counter takes $O(\log(n))$ cells. Then compares them.

Example 6.6. Recall $PATH = \{\langle G, s, t\rangle \mid G$ is a directed graph and there is a path $s \to t\} \in$ P. We show $PATH \in NLOGSPACE$ by guessing a path. We only need to "remember" on which node we stand in any given moment moment. We also maintain a counter i until $|V|$ (once again, in binary). $\mathcal{M}$ operates the following way:

1. Write on the strip $s_0 = s, i = 0$.
2. While $i \leq |V|$, if $s_i = t$ then halt and accept. Otherwise - guess a neighbor $s_i \to s_{i+1}$, and $i++$.
3. Halt and reject.

Correctness is clear: There is a path $s \to t$ iff there is a simple path $s \to t$ iff $\mathcal{M}$ might guess this path iff $\mathcal{M}$ accepts.

Is $PATH \in LOGSPACE$? By Savitch. $\text{NSPACE}(s(n)) \subset \text{SPACE}(s^2(n))$, but this does not imply $NLOGSPACE \subset LOGSPACE$

Completeness and Hardness

The definitions are the same - with a twist: Polynomial reductions are not good enough[IV] - so we need a new notion of reduction:

[IV] We will see that $NL \subset$ PTIME

Definition 6.8 (Logspace Reduction)**.** A function $f : \Sigma^* \to \Sigma^*$ is a **Logspace Reduction** if there exists a **Logspace transducer** $\mathcal{M}$. That is, f is computable by $\mathcal{M}$ which operates with 3 strips (input - RO, work - RW, and output - WO): The work strip is with $O(\log(|w|)$ cells, and $\mathcal{M}$ writes $f(w)$ on the output strip.

Remark. If f is such that $w \in \mathcal{A} \iff f(w) \in \mathcal{B}$, we write $\mathcal{A} \leq_L \mathcal{B}$.

Example 6.7. Consider a digraph with two types of nodes. An initial node s holds a token. The first type node ("first player") does not want the token to reach t, and the second type node ("second player") wants the token to reach t. This problem is in P and in fact is P-complete.

Theorem 6.7. *$PATH$ is NL-complete.*

Proof. It is left to show that for any $\mathcal{L} \in NL$, $\mathcal{L} \leq_L PATH$. The idea: let $\mathcal{M}$ be a logspace decider of $\mathcal{L}$. The graph G would be the configurations graph, $s = C^w_{init}$ and $t = C_{acc}$ the accepting configuration (as always, assume uniqueness). G's nodes would be all configurations, and edges iff subsequent configurations.
Any configuration is described by an element of $\Gamma^i(\Gamma \times Q)\Gamma^{s(n)-i+1}\#(0+1)^{\log(n)}$, which requires $2\log(n) + 1 = O(\log(n))$ space. The reduction go over all words over $\Gamma \cup (Q \times \Gamma) \cup \{\#, 0, 1\}$ and write those who describe a valid configuration. These are $V(G)$. For E, iterate over pairs of words and copy those who correspond to subsequent configurations. For s, t - we just write them down, as we know their structure. □

Remark. There were some results for which we demanded $s(n) \geq n$. This is because we bounded the number of configurations with $2^{ds(n)}$ for some d, and wanted to notice something exponential. Even when $s(n) = O(\log(n))$, there exists d such that the number of configurations is bounded by $2^{d \cdot s(n)}$: In this case, the number of configurations is $|Q| \cdot |\Gamma|^{s(n)} \cdot s(n) \cdot n$ with the n indicating the reading head's location in the input strip. When $s(n) = O(\log(n))$, the number of configurations is bounded by $k_1 \cdot k_2^{d_1 \log(n)}(d_1 \log(n))2^{\log(n)} \leq 2^{d \log(n)}$ for some d. Hence Savich holds even in logspace.

Theorem 6.8. *$NL \subset P$*

Proof. Let $\mathcal{L} \in NL$. Reduce $\mathcal{L} \leq_L PATH$ (takes polytime since the number of configurations of $\mathcal{L}$ is polynomial). Afterwards run DFS or BFS (in polytime) and respond accordingly. □