UiO : Department of Informatics
University of Oslo

Investigations in complexity theory related to the space hierarchy theorem

Andreas R. Askeland

Master's Thesis Autumn 2015

Investigations in complexity theory related to the space hierarchy theorem

Andreas R. Askeland

2nd November 2015

Abstract

This master thesis investigate space complexity theory, with the motivation of developing a degree theory. A direct application of our investigation is a strengthening of the space hierarchy theorem as stated by Sipser in [3]. At the end, we define a degree theory and show some elementary properties of the degree structure. We also present detailed proofs of known results, including a detailed description of a universal Turing machine.

Acknowledgements

I would not have been able to write this thesis without the help from my supervisor Lars Kristiansen. His directions into the field of complexity theory has made me really curious about this fascinating field, which has kept me intensely motivated. I would also like to thank Roger Antonsen, Herman Ruge Jervell, Arild Waaler, and Andreas Nakkerud for giving inspiring lectures in logic and computations. I have also received helpful feedback from Lars Kristian Maron Telle, Lars Tveito, Evgenij Thorstensen, and Sigurd Kittilsen, for which I am grateful. The love and support from Hilde Bakken Reistad, together with the rest of my family and my friends, made me thrive through all of this.

Contents

Chapter 0

Introduction

Many of the great open questions in complexity theory, like $P \neq PSPACE$ and $P \neq NP$, are statements about how time classes relates to space classes, or how deterministic classes relates to non-deterministic classes. We do not have many strong results about any of these relationships at the current moment. What we do have, is a strong result about how we can separate space classes from each other, thanks to the space hierarchy theorem. The time hierarchy theorem (see [3]) is a similar result about time complexity, but in this thesis we will focus on space complexity only. Intuitively, the space hierarchy theorem states that with more space, a machine may solve more problems. The standard version of this theorem were first given in 1965 by Hartmanis et al. in [4]. Sipser [3] restates it in the following way: for any space constructible functions g that is at least $O(log(n))$, there exists a language decidable in space $O(g(n))$, but not in space $f(n) \in o(g(n))$. One of our main goals is to strengthen this result for certain functions by weakening the criteria from $f(n) \in o(g(n))$ to $g(n) \notin O(f(n))$. There are similar results in the literature [1] which we believe our result follow from. Nevertheless, we have proven our results independently and in much more detail than what is usually done. Our main motivation is to present a theory of degree, analogous with Turing degrees in computability theory.

Our stronger version of the space hierarchy theorem states that for any two monotone functions g and f that are space constructible, and such that $f(n) \geq log_2(n+2) \leq g(n)$, if $g(n) \notin O(f(n))$, then there exist a language decidable in space $O(g(n))$ on a deterministic Turing machine, but not in space $O(f(n))$. The requirement $f(n) \in o(g(n))$ from the well known space hierarchy theorem is weaker than $g(n) \notin O(f(n))$ because there are functions $g \notin O(f(n))$ and $f(n) \notin o(g(n))$. An example is the monotone functions $f, g \colon \mathbb{N} \to \mathbb{N}$ defined by

$$f(n) = \begin{cases} 2 & \text{if } n = 0 \\ f(n-1)^2 & \text{if n is even and } n > 0 \\ f(n-1)+1 & \text{if n is odd and } n > 0 \end{cases}$$

$$g(n) = \begin{cases} 2 & \text{if } n = 0 \\ g(n-1)+1 & \text{if n is even and } n > 0 \\ g(n-1)^2 & \text{if n is odd and } n > 0 \end{cases}$$

These functions will never dominate each other. We can see their behavior in figure 1.

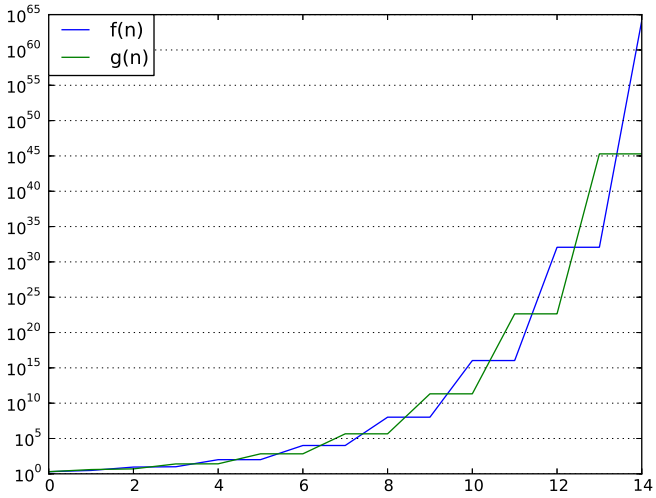

Figure 1: $g \notin O(f(n))$ and $f(n) \notin o(g(n))$

In his proof, Sipser claims that there exists a Turing machine D, such that when $w = \langle M \rangle 10^*$ is given as input, D can

1. Decide if the input is of the form $\langle M \rangle 10^*$, where $\langle M \rangle$ is a description of a Turing machine M.
2. Simulate M on w.
3. Halt if M uses more than $2^{g(|w|)}$ number of steps on w.
4. Halt if D uses more than $g(n)$ space.

According to Sipser's criteria, $g(n)$ may be $log(n)$. In this case, D is not allowed to use more than $log(n)$ space if M use $o(log(n))$ space. While it is true that such a Turing machine D exists, constructing one is not trivial. One challenge is that M and w have to be encoded into symbols of the fixed input alphabet of D, even though M may have a much larger input alphabet than D has. One symbol on the tape of M must therefore be represented by many symbols on the tape of D. In chapter 2, we will see how any Turing machine along with any input string can be encoded. We will measure space by how many cells on the work tape is read. Since D must have the ability to work in sublinear space, we let our Turing machines have a read-only input tape, and a separate work tape. Otherwise, D does not have enough space to read the entire input with

$log(n)$ as space requirement. Another challenge is how D can decide if $\langle M \rangle$ really encodes a Turing machine. According to Sipser's definition of Turing machines (modified by an extra tape), D must verify that the function

$$\delta \colon Q \times \Sigma \times \Gamma \to Q \times \Gamma \times \{L, R\} \times \{L, R\}$$

is totally defined in $\langle M \rangle$. This means that, for each state $q \in Q$, each symbol $c_1 \in \Sigma$, and for each tape symbol $c_2 \in \Gamma$, the tuple $((q, c_1, c_2), (q', c', D, D'))$ must be represented in $\langle M \rangle$. Then, we have to construct the simulation part of D, which means that D must have built in the property of being a universal Turing machine. The encoding of M and w must therefore be of a form such that D is able to carry out each step of M on w from this description. Similar with what Alan Turing did in 1936 [5], we present a universal Turing machine $\mathcal{U}$ in chapter 3, which will use our encoding from chapter 2. D also has to halt if it tries to read more than $g(n)$ number of cells on the work tape. Since g is space constructible, D can mark off $g(n)$ number of cells on the work tape, and reject if it needs more space. Nevertheless, it complicates the simulation a bit, since M may reach one of the markers before having read $g(n)$ number of cells. In this case, D must shift every symbol on the work tape as a part of the simulation. Finally, D has to count each step of M during the simulation and halt if the counter ever becomes $2^{g(|w|)}$. If $g(|w|) = log(|w|)$, D must be able to keep all this information on the work tape, including the configuration of M, and still run in space $log(|w|)$. In chapter 4, we define a similar universal Turing machine $\mathcal{U}_f$, which is based on $\mathcal{U}$, but rejects if the simulation requires more than a certain amount of space, or if the Turing machine being simulated requires more than a certain amount of steps. In chapter 5, and in appendix C, we present the Turing machine D_g, which is based on $\mathcal{U}_f$. D_g will be constructed for the purpose of diagonalization, which plays the same role in our proof of the stronger space hierarchy theorem as the Turing machine D in Sipser's proof. D_g will be constructed so that on some specific input, it will reject if and only if $\mathcal{U}_f$ accepts a specific input. In comparison with the proof of the space hierarchy theorem given by Sipser, and in the original proof by Hartmanis et al., we give a lot more details. We have chosen to do so because these proofs relies on many statements which we find interesting to investigate. In chapter 6, we present our main result, which is a stronger version of the space hierarchy theorem. We will see that our proof relies on most of the definitions and results given in the preceding chapters. As a consequence of our main result, we construct a theory of degree of space complexity. This theory will be the topic of chapter 7. We have chosen to describe many of the Turing machines in detail, but we have also moved some of the details into appendices to enhance the readers experience. In the preliminaries, which is the following chapter, we will present some results together with the most basic definitions which we will need throughout the rest of this thesis. Any reader familiar with complexity theory will find most of this chapter very familiar.

Chapter 1

Preliminaries

In this chapter, we introduce the basic notation we will need, such as languages, big-O, and the Turing machine as our computational model. We will be precise about what we mean by space. The problems we classify will be formulated as decision problems. A Turing machine M solves a problem within some space bound if for any string, M decides if the string is in a certain language without breaching the bound. This will be more precise after the following definitions.

Definition 1 (Alphabet) *An **alphabet** Σ is a non-empty finite set of symbols.*

Definition 2 (String) *A **string** is a finite sequence of symbols from an alphabet. We use $|s|$ to denote the length of s, and s^n to denote the string $\underbrace{s \dots s}_{n}$. The **empty string** ϵ has length zero.*

We will often use α, β, and σ when representing strings.

Definition 3 (Σ^*) *Σ^* is the set of all strings over Σ.*

This means that if $\Sigma = \{0, 1\}$, then $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, ...\}$.

Definition 4 (Language) *Let Σ be an alphabet. A **language** is a subset of Σ^*.*

We will work with Turing machines having one work tape and one read-only input tape. A Turing machine consists of a finite state control, a read-only input tape, and an infinite read/write work tape. The input tape is finite, with the $\triangleright$ symbol at the left end, and the $\llcorner$ symbol at the right end. At the start of the computation, the input head is at the symbol to the right of $\triangleright$, which is $\llcorner$ if the input is ϵ. The work tape is filled with infinitely many $\llcorner$ symbols. Depending on which state the Turing machine is in, which symbol it reads from the input tape, and which symbol it reads from the work tape, the finite state control defines how it changes state, which symbol it writes on the work tape, and how each head move one step to the left, right, or stay. The Turing machine continues until it reaches an accept or reject state, or it might not halt at all. We use the following definition inspired by [3] and [2].

Definition 5 (Turing machine) *A **Turing machine** is a 7-tuple* $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ *where*

- Q *is a non-empty finite set of* ***states***
- Γ *is the non-empty finite* ***work tape alphabet*** *containing the* ***blank symbol*** $\sqcup$
- $\Sigma \subseteq \Gamma$ *is the* ***input alphabet****, and does not contain* $\triangleright$ *and* $\sqcup$
- $q_0 \in Q$ *is the* ***start state***
- $q_{accept} \in Q$ *is the* ***accept state***
- $q_{reject} \in Q$ *is the* ***reject state*** *and* $q_{accept} \neq q_{reject}$.
- *The* ***transition function***

  $$\delta \colon Q \setminus \{q_{accept}, q_{reject}\} \times (\Sigma \cup \{\triangleright, \sqcup\}) \times \Gamma \to Q \times \Gamma \times \{L, R, S\} \times \{L, R, S\}$$

  such that for any q, s, q', s', D

  $$\delta(q, \triangleright, s) \neq (q', s', L, D)$$
  $$\delta(q, \sqcup, s) \neq (q', s', R, D)$$

The following definition clarifies what it means for a Turing machine to decide a language. We call such Turing machines *deciders*.

Definition 6 (Decidable language) *A turing machine* M ***decides*** *a language* A *if and only if* M *on any input* α *halts in* q_{accept} *if* $\alpha \in A$ *and* q_{reject} *if* $\alpha \notin A$.

The notions of *space* and *space bound* is defined in the following way.

Definition 7 (Space) *Let* M *be a Turing machine. We define the function* $S_M \colon \Sigma^* \to \mathbb{N}$

$S_M(\alpha) =$ *the number of distinct cells on the work tape scanned by* M *on input* α

Definition 8 (Space bound) *Let* $f \colon \mathbb{N} \to \mathbb{N}$ *be a function. A Turing machine* M *has* ***space bound*** f*, if for any input* α*,* M *halts and*

$$S_M(\alpha) \leq f(|\alpha|)$$

We now introduce the relation $\leq_O$, which is equivalent to the more common big-O notation. The degree theory which we present in chapter 7 will be based on this relation. Our motivation for introducing $\leq_O$ is to make our degree theory neat.

Definition 9 ($\leq_O$) *Let* $f, g \colon \mathbb{N} \to \mathbb{R}^+$ *be functions, where* $\mathbb{R}^+$ *is the set of non-negative real numbers. The relation* $f \leq_O g$ *holds if there exist* $c, n_0 \in \mathbb{N}$ *such that for all* $n \geq n_0$ *we have that*

$$f(n) \leq cg(n)$$

By $f \not\leq_O g$ we mean that it is not the case that $f \leq_O g$, which means that for any $c, n_0 \in \mathbb{N}$, there exist $n \geq n_0$ such that

$$f(n) > cg(n)$$

Let us verify that $\leq_O$ is reflexive and transitive.

Lemma 10 *$\leq_O$ is reflexive.*

Proof. Let $c = 1$. Then $f(n) \leq cf(n)$ for all n. Thus $f \leq_O f$. □

Lemma 11 *$\leq_O$ is transitive.*

Proof.

- Assume that $f \leq_O g$ and $g \leq_O h$.
- Then there exist $c, n_0 \in \mathbb{N}$ such that $f(n) \leq cg(n)$ for $n \geq n_0$, and $c', n_0' \in \mathbb{N}$ such that $g(n) \leq c'h(n)$ for $n \geq n_0'$.
- Then $f(n) \leq cc'h(n)$ for $n \geq max(n_0, n_0')$
- Thus $f \leq_O h$.

□

When we later present the space hierarchy theorem as stated by Sipser, we need the definition of big O.

Definition 12 (Big-O) *[3] Let f and g be functions $f, g \colon \mathbb{N} \to \mathbb{R}^+$. Say that $f(n) = O(g(n))$ if positive integers c and n_0 exist such that for every integer $n \geq n_0$*

$$f(n) \leq cg(n)$$

Lemma 13 *$f(n) = O(g(n))$ if and only if $f \leq_O g$*

Proof. $f(n) = O(g(n))$ is defined exactly as $f \leq_O g$. □

Often we will need a Turing machine computing some function. The notion of space constructible function is therefore usefull. The following is an equivalent definition compared with how Sipser defines space constructible.

Definition 14 (Space constructible function) *A function $f \colon \mathbb{N} \to \mathbb{N}$ such that $log \leq_O f$, is **space constructible** if there exits a Turing machine M with space bound $g \leq_O f$ such that on any input α of length n, M halts with $f(n)$ on the tape written in binary.*

As mentioned in the introduction, our stronger version of the space hierarchy theorem will hold for certain functions. These are the functions which we will call *honest*.

Definition 15 (Honest function) *A function $f : \mathbb{N} \to \mathbb{N}$ is **honest** if*

(1) $f(n) \leq f(n+1)$ *(monotone)*

(2) $\log_2(n+2) \leq f(n)$

(3) *f is space constructible.*

The following lemma gives an alternative defintion of $\leq_O$ for honest functions.

Lemma 16 *Let g and f be honest functions. Then, $f \leq_O g$ if and only if there exists $a, b \in \mathbb{N}$ such that for all n*

$$f(n) \leq ag(n) + b$$

Proof.

- To prove the right direction, assume $f \leq_O g$.
- Then there exist $c, n_0 \in \mathbb{N}$ such that for all $n \geq n_0$ we have that

$$f(n) \leq cg(n)$$

- Let $a = c$ and $b = \max_{x \leq n_0} f(x)$. Then it follows that for all n

$$f(n) \leq ag(n) + b$$

- To prove the left direction, assume that there exist a, b such that for all n

$$f(n) \leq ag(n) + b$$

- Since g is honest, $\log_2(n+2) \leq g(n)$ which implies $g(n) \geq 1$. Thus $bg(n) \geq b$ for all n.
- Let $c = a + b$. Then there exist $n_0 \in \mathbb{N}$, for instance $n_0 = 0$, such that for all $n \geq n_0$ we have that

$$f(n) \leq cg(n)$$

□

The following lemma will be useful later when we need some space bound to be honest. We will see the term $\lfloor log_2(n) \rfloor + 1$ in the proof of this lemma, and in many other of our proofs later, to give us the length of the binary representation of some number n.

Lemma 17 *Let M be a Turing machine with space bound $g \leq_O f$ and let f be honest. Then there exist an honest function g' such that M has space bound $g' \leq_O f$.*

Proof.

- Let M be a Turing machine with space bound $g \leq_O f$ and let f be honest.
- By the definition of $\leq_O$, there exist $c, n_0 \in \mathbb{N}$ such that for all $n \geq n_0$ we have that $g(n) \leq cf(n)$.

- Let $g'(n) = cf(n)$ for some $c \geq 1$ such that $g(n) \leq cf(n)$.
- The following shows that g' is honest:

  (1) $g'(n) \leq g'(n+1)$ holds since $cf(n) \leq cf(n+1)$.

  (2) $log_2(n+2) \leq g'(n)$ holds since $log_2(n+2) \leq cf(n)$.

  (3) Since f is space constructible, there exist a Turing machine N with space bound $h \leq_O f$ such that on any input α of length n, N halts with $f(n)$ on the tape written in binary. Let N' use N to compute $f(n)$, and then compute $g'(n)$ by adding $f(n)$ to a copy of itself c times. During this computation the work tape will be of the form

  $$\llcorner cf(n)\$f(n)\$f(n)\lrcorner$$

  where $f(n)$ and $cf(n)$ is written in binary. Notice that since f is honest we have that $f(n) \geq 1$, $f(n) > log_2(n)$, and since $h \leq_O f$ there exist l, m_0 such that $h(m) \leq lf(m_0)$ for $m \geq m_0$. Thus N' has space bound

  $$\begin{aligned} h'(m) &= h(m) + \lfloor log_2(cf(m)) \rfloor + 1 + 2\lfloor log_2(f(m)) \rfloor + 1 + 4 \\ &\leq lf(m) + log_2(c) + 3log_2(f(m)) + 6 \\ &< (l + log_2(c) + 3 + 6)f(m) \\ &\text{for all } m \geq m_0 \end{aligned}$$

  This shows that N' has space bound $h' \leq_O f$ such that on any input α of length n, N' halts with $g'(n)$ on the tape written in binary. It follows that g' is space constructible.
- Thus M has space bound $g' \leq_O f$ and g' is honest.

□

Classifying problems of similar complexity is fundamental for complexity theory, and the common way to do this with focus on space complexity is given by the following definition.

Definition 18 (Space complexity class) *Let $f, g\colon \mathbb{N} \to \mathbb{R}^+$. A **space complexity class** $SPACE(g)$ is the set of all languages decidable by a Turing machine with space bound f such that $f \leq_O g$.*

Chapter 2

Encoding Turing machine and input

We will soon define a universal Turing machine $\mathcal{U}$, but first we need to specify the input format for which we will construct $\mathcal{U}$ for, which is exactly the topic of this chapter. The following will specify how any Turing machine M, along with any string α over the work tape alphabet of M can be encoded. Notice that this also defines an encoding for any input string given to M, since the work tape alphabet contains the input alphabet for any Turing machine (see definition 5). One challenge is that an input string α can be a string over any finite language, while the input language of $\mathcal{U}$ has to be fixed. Another challenge is how to represent an arbitrary M so that $\mathcal{U}$ can perform the instructions of M from this description. The following defines an *encoding* for M and α, which we denote by

$$\langle M \rangle \langle \alpha \rangle$$

Definition 19 (Encoding) *We let $\langle c \rangle$ denote the encoding of the symbol c. For any string $s = c_1 \dots c_n$, we let $\langle s \rangle = \langle c_1 \rangle \dots \langle c_n \rangle$. The exception is when we use the letters α, β, or σ. If $\alpha = \alpha_1, ..., \alpha_n$, $\beta = \beta_1, ..., \beta_n$, and $\sigma = \sigma_1, ..., \sigma_n$ we let*

$$\langle \alpha \rangle \equiv \langle \triangleright \alpha_1 \dots \alpha_n \sqcup \rangle$$

$$\langle \beta \rangle \equiv \langle \triangleright \beta_1 \dots \beta_n \sqcup \rangle$$

$$\langle \sigma \rangle \equiv \langle \triangleright \sigma_1 \dots \sigma_n \sqcup \rangle$$

We also define the following encoded symbols

$$\langle \epsilon \rangle \equiv \epsilon$$

$$\langle \sharp \rangle \equiv [0^1]$$

$$\langle (\rangle \equiv [0^2]$$

$$\langle) \rangle \equiv [0^3]$$

$$\langle , \rangle \equiv [0^4]$$

$$\langle \rightarrow \rangle \equiv [0^5]$$

$$\langle L \rangle \equiv [0^6]$$
$$\langle R \rangle \equiv [0^7]$$
$$\langle S \rangle \equiv [0^8]$$

If M is a Turing machine with Q, Σ, and Γ, as the set of states, the input alphabet, and the work tape alphabet respectively, we define the encoding of M to be the encoded start state, followed by the transition function δ.

$$\langle M \rangle \equiv \langle q_0 \delta \rangle = \langle q_0 (q, c, d) \to (q', d', D, D') \ldots \rangle$$

where $q, q' \in Q$, $c \in \Sigma$, $d, d' \in \Gamma$, and $D, D' \in \{L, R, S\}$.

$$\langle \triangleright \rangle \equiv [1]$$
$$\langle \sqcup \rangle \equiv [10]$$
$$\langle q_{accept} \rangle \equiv [11]$$
$$\langle q_{reject} \rangle \equiv [100]$$

We assign a binary number to each of the remaining states and symbols of the tape alphabet. Let $\Gamma \setminus \{\sqcup\} = \{c_1, \ldots, c_m\}$. We assign a unique number $i \in \{5, \ldots, m+4\}$ to each symbol in $\Gamma \setminus \{\sqcup\}$, and let b equal i written in binary.

$$\langle c_i \rangle \equiv [b]$$

Let $Q \setminus \{q_{accept}, q_{reject}\} = \{q_1, \ldots, q_n\}$. We assign a unique number $i \in \{m+5, \ldots, m+4+n\}$ to each of the states in $Q \setminus \{q_{accept}, q_{reject}\}$. The encoding for the i'th state is defined by

$$\langle q_i \rangle \equiv [b]$$

where b is i written in binary. If $\langle c \rangle = [b]$ is any encoding of a state or tape symbol, the marked version $\dot{c}$ is encoded as

$$\langle \dot{c} \rangle \equiv [0b]$$

We will use π to denote a pointer and let

$$\langle \pi \rangle \equiv [\pi]$$

For any input $\beta = \beta_1 \ldots \beta_n$, such that $\beta \in \Sigma^*$, π is the binary number corresponding to the position in $\langle \beta \rangle$ which is obtained by counting each of the symbols

$$\triangleright \beta_1 \ldots \beta_n \sqcup$$

from right to left π times. We will often refer to π as the counter.

As we can see from this definition, some symbols have the same encoding for any Turing machine, while others are dependent on which Turing machine we are encoding. When assigning a unique number to each state and each symbol from an alphabet it will not matter which enumeration we choose. Lets look at an example.

Example 20 *Let M be a Turing machine with $Q = \{q_0, q_{accept}, q_{reject}\}$, $\Sigma = \{0, 1, a\}$, $\Gamma = \Sigma \cup \{\textvisiblespace\}$, and transition function δ defined by the table below.*

$q_0, \triangleright, \textvisiblespace$	$\rightarrow$	$q_0, \textvisiblespace, R, S$
$q_0, \triangleright, 0$	$\rightarrow$	$q_0, \textvisiblespace, R, S$
$q_0, \triangleright, 1$	$\rightarrow$	$q_0, \textvisiblespace, R, S$
$q_0, \triangleright, a$	$\rightarrow$	$q_0, \textvisiblespace, R, S$
$q_0, 0, \textvisiblespace$	$\rightarrow$	$q_0, \textvisiblespace, R, S$
$q_0, 0, 0$	$\rightarrow$	$q_0, \textvisiblespace, R, S$
$q_0, 0, 1$	$\rightarrow$	$q_0, \textvisiblespace, R, S$
$q_0, 0, a$	$\rightarrow$	$q_0, \textvisiblespace, R, S$
$q_0, 1, \textvisiblespace$	$\rightarrow$	$q_0, \textvisiblespace, R, S$
$q_0, 1, 0$	$\rightarrow$	$q_0, \textvisiblespace, R, S$
$q_0, 1, 1$	$\rightarrow$	$q_0, \textvisiblespace, R, S$
$q_0, 1, a$	$\rightarrow$	$q_0, \textvisiblespace, R, S$
$q_0, a, \textvisiblespace$	$\rightarrow$	$q_{reject}, 0, R, S$
$q_0, a, 0$	$\rightarrow$	$q_{reject}, 0, R, S$
$q_0, a, 1$	$\rightarrow$	$q_{reject}, 0, R, S$
q_0, a, a	$\rightarrow$	$q_{reject}, 0, R, S$
$q_0, \textvisiblespace, \textvisiblespace$	$\rightarrow$	$q_{accept}, 1, S, S$
$q_0, \textvisiblespace, 0$	$\rightarrow$	$q_{accept}, 1, S, S$
$q_0, \textvisiblespace, 1$	$\rightarrow$	$q_{accept}, 1, S, S$
$q_0, \textvisiblespace, a$	$\rightarrow$	$q_{accept}, 1, S, S$

Following the definition above, we encode M and the input $\alpha = 01a$ as

$$\langle M \rangle \langle \alpha \rangle = \langle q_0 \delta \rangle \langle \triangleright 01 a \textvisiblespace \rangle$$

Writing out some of δ, we have

$$\langle q_0 (q_0, \triangleright, \textvisiblespace) \rightarrow (q_0, \textvisiblespace, R, S)$$

$$\vdots$$

$$(q_0, \textvisiblespace, a) \rightarrow (q_{accept}, 1, S, S) \rangle$$

$$\langle \triangleright 01 a \textvisiblespace \rangle$$

Encoding the states and the tape symbols, we get

$$[1000]([1000], [1], [10]) \rightarrow ([1000], [10], R, S)$$

$$\vdots$$

$$([1000], [10], [111]) \to ([11], [110], S, S)$$
$$[1][101][110][111][10]$$

Encoding all the special symbols, we get the final encoding
[1000][00][1000][0000][1][0000][10][000][00][1000][0000][10][0000][0000000]
[0000][00000000][000] . . . [00][1000][0000][10][0000][111][000][00000][00]
[11][0000][110][0000][00000000][0000][00000000][000][1][101][110][111][10]
This is the expected form of the input given to $\mathcal{U}$.

Later on, it will be useful to have a space bound for encoded strings, encoded states, and the pointer π, which the following three lemmas provide.

Lemma 21 *For any Turing machine M, with work tape alphabet Γ, there exist $k \in \mathbb{N}$ such that for all $s \in \Gamma^*$*

$$|\langle s \rangle| \leq k|s|$$

Proof.

- Let M be any Turing machine, and let Q be the set of states of M.
- Let c be an arbitrary symbol in Γ. By definition 19, c will be encoded as $\langle c \rangle = [b]$ where b is a binary number and $b \leq |\Gamma| + 4$. It follows that
$$|\langle c \rangle| \leq \lfloor log_2(|\Gamma| + 4) \rfloor + 1 + 2 = k$$
for some k given by M.
- Since no symbol in s takes more than k symbols to encode,
$$|\langle s \rangle| \leq k|s|$$

□

Lemma 22 *For any Turing machine M, with Q as set of states, there exist $k \in \mathbb{N}$ such that for all $q \in Q$*

$$|\langle q \rangle| \leq k$$

Proof.

- Let M be any Turing machine with work tape alphabet Γ, and let Q be the set of states of M.
- Let q be an arbitrary state in Q. By definition 19, q will be encoded as $\langle q \rangle = [b]$ where b is a binary number and $b \leq |\Gamma| + 4 + |Q|$. Then
$$|\langle q \rangle| \leq \lfloor log_2(|\Gamma| + 4 + |Q|) \rfloor + 1 + 2 = k$$
for some k given by M.

□

Lemma 23 *Let f be any honest function, M be any Turing machine with input alphabet Σ, and β be any string such that $\beta \in \Sigma^*$. For any pointer π representing a position in β*

$$|\langle \pi \rangle| < f(|\beta|) + 5$$

Proof.

- Let f be any honest function, M be any Turing machine with input alphabet Σ, and β be any string such that $\beta \in \Sigma^*$.
- By definition 19, π is a binary number corresponding to a position in $\langle \beta \rangle$ which is obtained by counting each of the symbols

  $$\triangleright \beta_1 \dots \beta_n \sqcup$$

  from right to left π times. Therefore π is no larger than $|\beta| + 2$ and

  $$|\langle \pi \rangle| = |[\pi]| \leq \lfloor log_2(|\beta| + 2) \rfloor + 1 + 2 \leq log_2(|\beta|) + 5 < f(|\beta|) + 5$$

  The last inequality holds since f is honest and therefore

  $$\log_2(n + 2) \leq f(n)$$

□

Chapter 3

Universial Turing machine

The goal of this chapter is to show that there exist a universal Turing machine $\mathcal{U}$ that can simulate any other Turing machine. We will define $\mathcal{U}$ so that our proof of the stronger space hierarchy theorem in chapter 6 will have a solid foundation. Although the existence of such a machine has been known since Alan Turing presented it in 1936 [5], we will define it for the encoding we saw in the last chapter, and for Turing machines having separate input and output tapes. The input format we have in mind when we construct $\mathcal{U}$, is $\langle M \rangle \langle \alpha \rangle$, where $\langle M \rangle$ is the encoding of some Turing machine M, and $\langle \alpha \rangle$ encodes some input α. $\mathcal{U}$ may be defined in many ways, and we will not attempt to define it in any optimal sense. We are ready to define our universal Turing machine $\mathcal{U}$.

Definition 24 (Universal Turing machine) *Let $\mathcal{U}$ be a Turing machine with input alphabet $\Sigma = \{0, 1, [,]\}$, and work tape alphabet $\Gamma = \Sigma \cup \{*, \llcorner\}$. The detailed description of how $\mathcal{U}$ works is given by the following pages.*

For input of the form $\langle M \rangle \langle \alpha \rangle$, where M is any Turing machine and α is any string over the input alphabet of M, $\mathcal{U}$ will simulate M on α. First, $\mathcal{U}$ initializes the work tape. Then it checks if M is in a halting state. If not, it simulates one step of M and do the halting check again. It repeats these two stages and halts if M halts. We have left out some of the details, but they are covered in appenix A. A challenge is what $\mathcal{U}$ should do on input which is not an encoded Turing machine. In this case, we have chosen that it will not be necessary for $\mathcal{U}$ to validate the input, although it rejects if the input is *too wrong* to be simulated. By *too wrong* we mean that if $\triangleright$ or $\llcorner$ is read from the input tape or the work tape at any time, except when $\mathcal{U}$ is described to search for $\triangleright$ or $\llcorner$, we let $\mathcal{U}$ reject. As a consequence, $\mathcal{U}$ may accept on input which is not a correctly encoded Turing machine. E.g., an encoded Turing machine with a transition rule left out that is not applied during the simulation. The following is a detailed description of how $\mathcal{U}$ works. The tape on top represents the input tape, and the tape underneath represents the work tape. Colored symbols indicate the head positions of $\mathcal{U}$.

Initialize work tape

We let the work tape head moves one step to the right for each symbol written. Initially, we expect the tapes to be of the following form.

$$\triangleright\langle q_0\delta\rangle\langle\triangleright\alpha_1\ldots\alpha_n\textvisiblespace\rangle\textvisiblespace$$

$$\textvisiblespace$$

1. Copy $\langle q_0\rangle$ to the work tape. This is done by copying each symbol, from the current position until, and including,].

$$\triangleright\langle q_0\delta\rangle\langle\triangleright\alpha_1\ldots\alpha_n\textvisiblespace\rangle\textvisiblespace$$

$$\langle q_0\rangle\textvisiblespace$$

2. Compute a pointer to the symbol $\langle\triangleright\rangle$, which initially is the symbol under the input head of M. This is done by writing $[0]$, and then moving the input head rightwards until $\textvisiblespace$, and then left until $\langle\triangleright\rangle$, counting each [on the work tape as $[\pi]$.

$$\triangleright\langle\ldots\triangleright\,\alpha_1\ldots\rangle\textvisiblespace$$

$$\langle q_0\pi\rangle\textvisiblespace$$

3. Copy α_1 to the work tape. This is done by moving the input head rightwards until [, and then copying each symbol from the input tape to the work tape, including the first].

$$\triangleright\langle\cdots\triangleright\,\alpha_1\ldots\rangle\textvisiblespace$$

$$\textvisiblespace\langle q_0\pi\alpha_1\rangle\textvisiblespace$$

4. Write $\langle\dot{\sharp}\textvisiblespace\rangle$. Move the work tape head left until $\textvisiblespace$, and then one step to the right. Move the input head left until $\triangleright$, and then right past $\langle($. Move to the **halting check** described below.

$$\triangleright\langle q_0(q,c,d)\to(q',d',D,D')(\ldots\rangle\textvisiblespace$$

$$\langle q_0\pi\alpha_1\dot{\sharp}\textvisiblespace\rangle$$

Following example 20, the tapes of $\mathcal{U}$ will at this stage be

$$\triangleright\langle q_0(q_0,\triangleright,\textvisiblespace)\to(q_0,\textvisiblespace,R,S)\ldots(q_0,\textvisiblespace,a)\to(q_{accept},1,S,S)\rangle\langle\triangleright01a\textvisiblespace\rangle\textvisiblespace$$

$$\langle q_0\pi0\dot{\sharp}\textvisiblespace\rangle$$

where $\langle\pi\rangle = [100]$, which is equal to the decimal number 4. This means that the input head of M is currently at the fourth symbol of $\triangleright01a\textvisiblespace$, counting from the right. I.e the symbol 0.

Halting check

1. Read the current state at the work tape. If it is $[11]$, the current state is q_{accept}, and $\mathcal{U}$ accepts. If it is $[100]$, the current state is q_{reject}, and $\mathcal{U}$ rejects. Otherwise, go to the simulation stage below.

Simulation

One step of M will be simulated by $\mathcal{U}$ in the following way:

1. Compare the current state at the work tape with the state at the input tape. If they are not equal, move the input head to the next transition rule. This is accomplished by moving rightwards until the first symbol after $\langle)(\rangle$. Repeat this search until a state that matches the current state is found.

$$\triangleright\langle\ldots(q,c,d) \rightarrow (q',d',D,D')(\ldots\rangle_{\sqcup}$$
$$\langle q\pi c'\sharp\ldots\dot{e}\ldots\rangle$$

2. Move the input head past the first $\langle,\rangle$, and the work head past two] symbols.

$$\triangleright\langle\ldots(q,c,d) \rightarrow (q',d',D,D')(\ldots\rangle_{\sqcup}$$
$$\langle q\pi c'\sharp\ldots\dot{e}\ldots\rangle$$

   Compare c and c'. If they are not equal, move the input head rightwards to the first symbol after $\langle)(\rangle$. Move the work head to the leftmost non-blank symbol, and go to 1.

3. Move the input head past the first $\langle,\rangle$. Move rightwards on the work tape until $\langle\dot{e}\rangle$, which will be the only string of the form $[0v1w]$, such that $v,w \in \{0,1\}^*$.

$$\triangleright\langle\ldots(q,c,d) \rightarrow (q',d',D,D')(\ldots\rangle_{\sqcup}$$
$$\langle q\pi c\sharp\ldots\dot{e}\ldots\rangle$$

   If d does not equal e, move the input head rightwards to the first symbol after $\langle)(\rangle$. Move the work head to the leftmost non-blank symbol, and go to 1.

4. Now that $e = d$ the tapes looks like this:

$$\triangleright\langle\ldots(q,c,d) \rightarrow (q',d',D,D')(\ldots\rangle_{\sqcup}$$
$$\langle q\pi c\sharp\ldots\dot{d}\ldots\rangle$$

   Move the input head past the next $\langle,\rangle$ and replace $\langle\dot{d}\rangle$ by $\langle\dot{d'}\rangle$. After this replacement, we have lost track of the tape head. We will see why this happens in appendix A. Therefore, go left until $\sqcup$, and then right until $[0v1w]$, for $v,w \in \{0,1\}^*$.

$$\triangleright\langle\ldots(q,c,d) \rightarrow (q',d',D,D')(\ldots\rangle_{\sqcup}$$
$$\langle q\pi c\sharp\ldots\dot{d'}\ldots\rangle$$

5. Move the input head past two $\langle, \rangle$. If D' is $\langle S \rangle$, go to 6. If D' is $\langle L \rangle$, move the work head leftwards to the first [.

   If $\sharp$ is read, we need more tape space.

   $$\triangleright\langle \ldots (q, c, d) \to (q', d', D, L)(\ldots\rangle_\sqcup$$

   $$\langle q\pi c\sharp \dot{d}' \ldots\rangle$$

   Move the work head left until $\sqcup$. Move one step to the right, then move each symbol three squares (the length of $\langle\sharp\rangle$) to the left including the first $\sharp$.

   $$\langle q\pi c\sharp\sharp \dot{d}' \ldots\rangle$$

   Move the work head to the $\langle\sharp\rangle$ to the right, and replace it with $\langle\sqcup\rangle$

   $$\langle q\pi c\sharp\sqcup \dot{d}' \ldots\rangle$$

   We can now continue as we have enough tape space.

   $$\langle q\pi c\sharp \ldots f \dot{d}' \ldots\rangle$$

   Replace $\langle f\rangle$ with $\langle \dot{f}\rangle$. Move rightwards until the first [and replace $\langle \dot{d}'\rangle$ by $\langle d'\rangle$. After erasing a 0 from $\langle \dot{d}'\rangle$ we don't know where the tape head might be, but that does not matter here.

   $$\triangleright\langle \ldots (q, c, d) \to (q', d', D, L)(\ldots\rangle_\sqcup$$

   $$\langle q\pi c\sharp \ldots \dot{f} d' \ldots\rangle$$

   The case where D' is $\langle R\rangle$ is similar.

   $$\triangleright\langle \ldots (q, c, d) \to (q', d', D, R)(\ldots\rangle_\sqcup$$

   $$\langle q\pi c\sharp \ldots \dot{d}' \ldots\rangle$$

   Move the work head right until [. If $\sqcup$ is read at this position, replace it with $\langle\sqcup\rangle$

   $$\langle q\pi c\sharp \ldots \dot{d}'\sqcup\rangle$$

   Regard the symbol to the right of $\dot{d}'$ as g. Replace g with $\dot{g}$, and replace $\dot{d}'$ with d'.

   $$\triangleright\langle \ldots (q, c, d) \to (q', d', D, R)(\ldots\rangle_\sqcup$$

   $$\langle q\pi c\sharp \ldots d'\dot{g} \ldots\rangle$$

6. Move the input head leftwards until (, then one step to the right. Move the work head to the leftmost non-blank symbol, which is the first symbol of $\langle q\rangle$. Replace $\langle q\rangle$ with $\langle q'\rangle$.

   $$\triangleright\langle \ldots (q, c, d) \to (q', d', D, D')(\ldots\rangle_\sqcup$$

   $$\langle q'\pi c\sharp \ldots \dot{e} \ldots\rangle$$

7. Move the work head rightwards and past the symbol]. Move the input head rightwards and past two $\langle, \rangle$

$$\triangleright\langle \ldots (q, c, d) \to (q', d', D, D')(\ldots\rangle_\sqcup$$

$$\langle q' \pi c \sharp \ldots \dot{e} \ldots \rangle$$

If D is $\langle S \rangle$, go to 8. If D is $\langle L \rangle$, move the input head rightwards until $\sqcup$. Then, move leftwards while decrementing π for each [symbol, until $\pi = 0$. Move left to the next [symbol.

$$\triangleright\langle \ldots\ldots x c y \ldots\rangle_\sqcup$$

$$\langle q' 0 c \sharp \ldots \dot{e} \ldots \rangle$$

Move the work head past] and replace c by x.

$$\triangleright\langle \ldots\ldots x c y \ldots\rangle_\sqcup$$

$$\langle q' 0 x \sharp \ldots \dot{e} \ldots \rangle$$

After the replacement, we have lost track of the work tape head. Therefore, move to the leftmost non-blank symbol, and then rightwards past one] symbol, so that it is over the counter (i.e., the symbol 0 next to q'). Replace this counter with 1. Move the input head rightwards until $\sqcup$ while incrementing the counter for each [symbol.

$$\triangleright\langle \ldots h \rangle_\sqcup$$

$$\langle q' \pi' x \sharp \ldots \dot{e} \ldots \rangle$$

The case where D is $\langle R \rangle$ is similar, replacing c by y instead, and counting from y instead of x.

8. Move the input head left until $\triangleright$ and then rightwards and past the first $\langle($. Move the work head to the leftmost non-blank symbol and go to the **halting check**.

This is how $\mathcal{U}$ is meant to work, but some details were left out. In appenix A we can see how c is replaced with c' by $\mathcal{U}$ in the middle of the tape when $|\langle c' \rangle| \neq |\langle c \rangle|$, and how $\mathcal{U}$ increments a binary number.

$\mathcal{U}$ is a universal Turing machine

We will now state the main result of this chapter.

Theorem 25 *For any Turing machine M with input alphabet Σ, and $\beta \in \Sigma^*$,*

$$M \text{ accepts } \beta \Leftrightarrow \mathcal{U} \text{ accepts } \langle M \rangle \langle \beta \rangle$$

Proof. This theorem is a direct consequence of definition 19, together with the detailed description on how $\mathcal{U}$ works, as given above. □

The following lemma give a space bound for $\mathcal{U}$ relative to the Turing machine it is simulating.

Lemma 26 *Let f be an honest functions. For any Turing machine M with space bound f, input alphabet Σ, and $\beta \in \Sigma^*$, there exist $a, b \in \mathbb{N}$ such that*

$$S_{\mathcal{U}}(\langle M \rangle \langle \beta \rangle) \leq af(|\beta|) + b$$

Proof.

- Let f be an honest function, M be a Turing machine with space bound f, Q be the set of states of M, and Γ be the work tape alphabet of M.
- From the definition of $\mathcal{U}$, the work tape of $\mathcal{U}$ will at any time be of the form $\langle q\pi x\sharp s\rangle$. $S_{\mathcal{U}}(\langle M \rangle \langle \beta \rangle)$ will therefore be no more than $|\langle q\pi x\sharp s\rangle|$ for the longest $\langle q\rangle$, $\langle \pi\rangle$, $\langle x\rangle$, $\langle \sharp\rangle$, and $\langle s\rangle$.
- By lemma 22 there exist $k' \in \mathbb{N}$ such that $|\langle q\rangle| \leq k'$ for any $q \in Q$.
- By lemma 23, $|\langle \pi\rangle| < f(|\beta|) + 5$ for any honest function f.
- Since x is some symbol in Σ, lemma 21 gives us that there exist $k'' \in \mathbb{N}$ such that $|\langle x\rangle| \leq k''$.
- $|\langle \sharp\rangle| = |[0^1]| = 3$.
- Let $s \in \Gamma^*$ be a longest string of the scanned cells when M executes on input β. By lemma 21 there exist $k \in \mathbb{N}$ such that $|\langle s\rangle| \leq k|s|$.
- Since M has space bound f, $S_M(\beta) \leq f(|\beta|)$.
- Since
$$|\langle s\rangle| \leq k|s| \text{ and } |s| = S_M(\beta) \leq f(|\beta|)$$
we have that
$$|\langle s\rangle| \leq kf(|\beta|)$$
- By summarising and using lemma 16
$$S_{\mathcal{U}}(\langle M \rangle \langle \beta \rangle) = |\langle q\pi x\sharp s\rangle|$$
$$\leq k' + (f(|\beta|) + 5) + k'' + 3 + kf(|\beta|)$$
$$= (2 + k)f(|\beta|) + (k' + k'' + 6)$$
- Let $a = (2 + k)$ and $b = (k' + k'' + 6)$ and the lemma holds.

□

Chapter 4

Universial f-space Turing machine

In this chapter we will see a space bounded universal Turing machine $\mathcal{U}_f$, which will halt on any input. For each honest function f the following definition yields a universal Turing machine with space bound relative to f.

Definition 27 (f-space Turing machine) *Let f be an honest function. Let $\mathcal{U}_f$ be a Turing machine with input alphabet $\Sigma = \{0, 1, [,], \$\}$ and work tape alphabet $\Gamma = \Sigma \cup \{*, \llcorner\}$. Let M be any Turing machine and α be any input to M. For input of the form*

$$0^a 1 0^b \$ \langle M \rangle \langle \alpha \rangle$$

$\mathcal{U}_f$ runs as follows:

1. *Verify that the input α is of the form*

   $$0^a 1 0^b \$ \ldots \langle \triangleright \rangle z \langle \llcorner \rangle$$

   for any $a, b \geq 0$ and $z = s_1 \ldots s_n$, where s_i is of the form $[x]$ such that $x \in \{0,1\}^*$ for $0 \leq i \leq n$. Reject if α is not of this form.

2. *Compute $af(|\alpha|) + b$ and mark this many tape cells for $\mathcal{U}$ to simulate M on input α.*

3. *Simulate M on α while counting down from $4^{af(|\alpha|)+b}$ for each step of M. If the counter ever get to 0, then* reject.

4. *If the simulation requires more space than marked, then* reject.

5. *If M accepts, then* accept. *If M rejects, then* reject.

We can examine the details on how $\mathcal{U}_f$ works in appendix B. The following lemma states that $\mathcal{U}_f$ is a decider.

Lemma 28 *$\mathcal{U}_f$ halts on any input $\sigma \in \{0, 1, [,], \$\}^*$.*

Proof.

- In stage 1., $\mathcal{U}_f$ just scans the input and either halts or goes to stage 2. after it reads a $\llcorner\!\lrcorner$ symbol.

- It is easily verifiable from the details in appendix B that the computation of $af(|\alpha|) + b$ will be completed so that $\mathcal{U}_f$ continues with stage 3.

- From appendix B, we see that $\mathcal{U}_f$ essentially executes $\mathcal{U}$ on $\langle M\rangle\langle\alpha\rangle$. From the detailed description of $\mathcal{U}$ in chapter 3, we find that the only places $\mathcal{U}$ may go back to a previous step in the algorithm, is in the *simulation stage*, step 2., 3., and 8. In step 2. and 3., $\mathcal{U}$ compare the current symbol at the position of the input head with the current symbol at the position of the work tape head. The input head move only in the right direction. If a sequence of symbols are equal, $\mathcal{U}$ goes to the next stage in the algorithm which is towards the 8'th step of the *Simulation stage*. Otherwise, if $\mathcal{U}$ read a $\llcorner\!\lrcorner$ symbol from the input tape, $\mathcal{U}$ will halt, causing $\mathcal{U}_f$ to halt as well. Otherwise, $\mathcal{U}$ goes back to stage 1., which only continues to move the input head in the rightward direction. We therefore see that in step 2. and 3. of $\mathcal{U}$, it will not be an infinite loop. Each time $\mathcal{U}$ reach stage 8., $\mathcal{U}_f$ decrease the counter (which started out as $4^{af(|\alpha|)+b}$), which eventually will get 0, causing $\mathcal{U}_f$ to halt. At any other step of $\mathcal{U}$, either $\mathcal{U}$ finds a symbol it is looking for, causing the algorithm to proceed towards the 8'th step of the *Simulation stage*, or the symbol is not found, causing $\mathcal{U}$ to halt, and $\mathcal{U}_f$ halts. Each time the 8'th step of the *Simulation stage* is reached, $\mathcal{U}_f$ decreases the counter, and continues $\mathcal{U}$ from the halting stage. Either $\mathcal{U}_f$ halts at the *Halting stage* or it start the simulation stage again.

- Thus, $\mathcal{U}_f$ will eventually halt.

□

The following theorem states that $\mathcal{U}_f$ is indeed the space bounded universal Turing machine we intended it to be.

Theorem 29 *Let f be an honest function. For any Turing machine M and $a, b \in \mathbb{N}$,*

$$\mathcal{U}_f \text{ accepts } 0^a10^b\$\langle M\rangle\langle\alpha\rangle$$

$$\Leftrightarrow$$

$$\mathcal{U} \text{ accepts } \langle M\rangle\langle\alpha\rangle \text{ and } S_{\mathcal{U}}(\langle M\rangle\langle\alpha\rangle) \leq af(|\alpha|) + b$$

Proof.

- Let f be an honest function.

- For the right direction, assume that $\mathcal{U}_f$ accepts $0^a10^b\$\langle M\rangle\langle\alpha\rangle$ for some $a, b \in \mathbb{N}$.

- By the definition of $\mathcal{U}_f$, $\mathcal{U}$ accepts $\langle M\rangle\langle\alpha\rangle$ and $\mathcal{U}$ can simulate M on α in space $af(|\alpha|) + b$, thus $S_{\mathcal{U}}(\langle M\rangle\langle\alpha\rangle) \leq af(|\alpha|) + b$.

- For the left direction, assume that $\mathcal{U}$ accepts $\langle M \rangle \langle \alpha \rangle$ and $S_{\mathcal{U}}(\langle M \rangle \langle \alpha \rangle) \leq af(|\alpha|) + b$ for some a, b.
- Since $\mathcal{U}_f$ halts on any input by lemma 28, it is sufficient to prove that $\mathcal{U}_f$ will not reject.
- Since the input $0^a 10^b \$ \langle M \rangle \langle \alpha \rangle$ is of the right form, $\mathcal{U}_f$ will not reject in stage 1.
- Observe that each configuration of M is represented on the work tape of $\mathcal{U}$ during the simulation. $\mathcal{U}$ can not have more than $4^{af(|\alpha|)+b}$ different work tapes of length $af(|\alpha|) + b$ since $\mathcal{U}$ has four symbols in the tape alphabet. Any M that can be simulated by $\mathcal{U}$ and halt with space bound $af(|\alpha|) + b$ must therefore halt within $4^{af(|\alpha|)+b}$ steps of M, otherwise $\mathcal{U}$ has entered a configuration of M twice and will loop forever. Since $\mathcal{U}$ accepts $\langle M \rangle \langle \alpha \rangle$, M accepts (and halts) on α by theorem 25. Since $S_{\mathcal{U}}(\langle M \rangle \langle \alpha \rangle) \leq af(|\alpha|) + b$, M can be simulated by $\mathcal{U}$ with space bound $af(|\alpha|) + b$. The counter will therefore not become 0.
- Since $S_{\mathcal{U}}(\langle M \rangle \langle \alpha \rangle) \leq af(|\alpha|) + b$, $\mathcal{U}$ can simulate M on α in space $af(|\alpha|) + b$. Thus $\mathcal{U}_f$ will not reject because the simulation requires more space than marked.
- Since M accepts α, it follows from the definition of $\mathcal{U}_f$ that $\mathcal{U}_f$ will not reject on $0^a 10^b \$ \langle M \rangle \langle \alpha \rangle$.
- Thus $\mathcal{U}_f$ accepts $0^a 10^b \$ \langle M \rangle \langle \alpha \rangle$.

□

Notice that the following similar assertion, where we have replaced *accepts* with *rejects*, is not true.

$$\mathcal{U}_f \text{ rejects } 0^a 10^b \$ \langle M \rangle \langle \alpha \rangle$$

$$\Leftrightarrow$$

$$\mathcal{U} \text{ rejects } \langle M \rangle \langle \alpha \rangle \text{ and } S_{\mathcal{U}}(\langle M \rangle \langle \alpha \rangle) \leq af(|\alpha|) + b$$

If we let M use no space, but loop forever, $\mathcal{U}_f$ will reject, while $\mathcal{U}$ will not halt. The following lemma will be useful in the proof of the stronger space hierarchy theorem, given as theorem 37 in chapter 6. It states that for suitable inputs, $\mathcal{U}_f$ decides the same language as the Turing machine M which it simulates.

Lemma 30 *Let f be an honest function and $L \in SPACE(f)$. Then, there exist $k \in \mathbb{N}$ and a Turing machine M deciding L such that for any $a, b \geq k$*

$$L \quad = \quad \{ \alpha \mid \mathcal{U}_f \textit{ accepts } 0^a 10^b \$ \langle M \rangle \langle \alpha \rangle \}$$

Proof.

- Let f be an honest function and $L \in \text{SPACE}(f)$.

- Since $L \in \text{SPACE}(f)$, there exists a Turing machine M with space bound $g \leq_O f$ deciding L. By lemma 17 there exist a honest function g' such that M has space bound $g' \leq_O f$.
- Since M has space bound g', it follows from lemma 26 that there exist a', b' such that for any α,

$$S_{\mathcal{U}}(\langle M \rangle \langle \alpha \rangle) \leq a' g'(|\alpha|) + b'$$

- Since g' and f is honest, and $g' \leq_O f$, it follows from lemma 16 that there exist $c, d \in \mathbb{N}$ such that

$$g'(|\alpha|) \leq cf(|\alpha|) + d$$

and equivalently

$$a' g'(|\alpha|) + b' \leq a' cf(|\alpha|) + da' + b'$$

- Let $k = max(a'c, da' + b')$. Then for any $a, b \geq k$, we have that

$$S_{\mathcal{U}}(\langle M \rangle \langle \alpha \rangle) \leq af(|\alpha|) + b \qquad (*)$$

- Now we have

$$\begin{aligned}
& \alpha \in L \text{ and (*)} \\
\Leftrightarrow\ & M \text{ accepts } \alpha \text{ and (*)} && (M \text{ decides } L) \\
\Leftrightarrow\ & \mathcal{U} \text{ accepts } \langle M \rangle \langle \alpha \rangle \text{ and (*)} && \text{(theorem 25)} \\
\Leftrightarrow\ & \mathcal{U}_f \text{ accepts } 0^a 1 0^b \$ \langle M \rangle \langle \alpha \rangle && \text{(theorem 29)}
\end{aligned}$$

□

Chapter 5

Diagonalising g-space Turing machine

We will here define the Turing machine D_g and prove some properties that will be usefull in the next chapter. A detailed description of how D_g works is given in appendix C. Her is a short definition. As with $\mathcal{U}_f$, the following definition yields a universal Turing machine with space bound relative to g for each honest function g.

Definition 31 (Diagonalizing g-space Turing machine) *Let g be an honest function. D_g is a Turing machine with input alphabet $\Sigma = \{0, 1, [,], \$\}$ and work tape alphabet $\Gamma = \Sigma \cup \{*, \sqcup\}$. Let M be any Turing machine. For input σ of the form*

$$0^a 1 0^b \$ \langle M \rangle$$

D_g runs as follows:

1. *Compute $g(|\sigma|)$ and mark this many tape cells for the simulation of M on input σ.*
2. *Simulate M on input σ while counting each step of M. If the counter ever exceeds $4^{g(|\sigma|)}$, then* accept.
3. *If the simulation requires more space than marked, then* accept.
4. *If M accepts,* reject. *If M rejects, then* accept.

The following lemma states that D_g is a decider.

Lemma 32 *D_g halts on any input $\sigma \in \{0, 1, [,], \$\}^*$.*

Proof.

- Observe that D_g is just a slightly different version of $\mathcal{U}_g$.
- It can easily be verified from the detailed description of D_g in appendix C that the behavior specific for D_g can not make it loop forever.

- This lemma therefore follow from lemma 28.

□

Since D_g is a decider, the next definition is justified. The language defined will be used for diagonalization in the proof of our stronger space hierarchy theorem 37 in the next chapter.

Definition 33 (The language L_{D_g}) *Let g be an honest function. The language L_{D_g} is the language decided by D_g.*

Lemma 34

$$L_{D_g} \in SPACE(g)$$

Proof.

- By definition, L_{D_g} is the language decided by D_g. By lemma 32, D_g halts for all input. It is therefore sufficient to prove that D_g has space bound f such that $f \leq_O g$. We also have that g is honest from the definition of D_g.

- From the details of D_g given in appendix C, $g(|\sigma|)$ is computed in less space than $h \leq_O g$. By lemma 17, there exist an honest function f' such that the computation of $g(|\sigma|)$ is space bound by $f' \leq_O g$. By lemma 16 there exist a, b such that

  $$f'(|\sigma|) \leq ag(|\sigma|) + b$$

  Thus our space bound so far is

  $$ag(|\sigma|) + b$$

- Then

  $$\text{␣}1\underbrace{0\ldots0}_{2g(|\sigma|)}\$\underbrace{\text{␣}\cdots\text{␣}}_{g(|\sigma|)}\$\text{␣}$$

  is computed with no more space than what is shown above. We now have that

  $$\begin{aligned} &|\text{␣}1\underbrace{0\ldots0}_{2g(|\sigma|)}\$\underbrace{\text{␣}\cdots\text{␣}}_{g(|\sigma|)}\$\text{␣}| \\ &= 1 + 1 + 2g(|\sigma|) + 1 + g(|\sigma|) + 1 + 1 \\ &\leq 3g(|\alpha|) + 5 \end{aligned}$$

- By definition of D_g, the simulation is done within the space shown above. Therefore,

  $$\begin{aligned} S_{D_g}(\alpha) &\leq ag(|\sigma|) + b + 3g(|\sigma|) + 5 \\ &\leq (a+3)g(|\sigma|) + (5+b) \end{aligned}$$

- Let $f(|\sigma|) = (a+3)g(|\sigma|) + (5+b)$. Then D_g has space bound f, and by lemma 16, $f \leq_O g$.

□

We finish this chapter by showing that on certain carefully constructed strings, D_g does the opposite of what $\mathcal{U}_g$ does for a similar input.

Lemma 35 *Let g be an honest function, M be a Turing machine, and*

$$\sigma = 0^a 1 0^b \$ \langle M \rangle$$

for some $a \geq 1$ and $b \geq 0$.

$$\mathcal{U}_g \text{ accepts } 0^1 1 0^0 \$ \langle M \rangle \langle \sigma \rangle$$

$$\Leftrightarrow$$

$$D_g \text{ rejects } \sigma$$

Proof.

- Let g be an honest function, M be a Turing machine, and

  $$\sigma = 0^a 1 0^b \$ \langle M \rangle$$

  for some $a \geq 1$ and $b \geq 0$.
- For the right direction, assume that $\mathcal{U}_g$ accepts $0^1 1 0^0 \$ \langle M \rangle \langle \sigma \rangle$.
- By definition 27, $\mathcal{U}_g$ simulates M on σ in space $g(|\sigma|)$, such that M accepts σ.
- By definition 31, D_g rejects σ.
- For the left direction, assume that D_g rejects σ.
- By definition 31, D_g have simulated M on σ in space $g(|\sigma|)$, such that M accepts σ.
- By definition 27, $\mathcal{U}_g$ accepts $0^1 1 0^0 \$ \langle M \rangle \langle \sigma \rangle$.

□

The details of how D_g does the opposite on input σ than what $\mathcal{U}_g$ does on input $0^1 1 0^0 \$ \langle M \rangle \langle \sigma \rangle$ can be found in appendix C.

Chapter 6

A stronger space hierarchy theorem

In this chapter we will see a stronger version of the space hierarchy theorem. We will also look at a known theorem called the space compression theorem. Then, we will see that for honest functions, the space hierarchy theorem as stated by Sipser [3] follows from our version of the theorem. Finally, we show that the stronger version of the space hierarchy theorem is indeed stronger. But first we need the following lemma.

Lemma 36 *Let f, g be honest functions such that $g \not\leq_O f$. Then, for any $a, b \in \mathbb{N}$, there exist infinitely many n such that*

$$g(n) > af(n) + b$$

Proof.

- Let g and f be honest functions and assume that $g \not\leq_O f$.
- Assume for contradiction that there exists a, b and n_0 such that for all $n \geq n_0$
$$g(n) \leq af(n) + b$$
- Let $c = g(n_0)$. Now $g(n) \leq af(n) + (b + c)$ for all n, since f and g are monotone. By lemma 16, $g \leq_O f$, which contradicts $g \not\leq_O$ f.

□

We are now ready to show our main result.

Theorem 37 (Stronger space hierarchy theorem) *Let g be honest. There exists a language L such that*

$$L \in SPACE(g)$$

and

$$L \notin SPACE(f)$$

for any honest f such that $g \not\leq_O f$.

Proof.

- By definition 33 of L_{D_g}, D_g decides L_{D_g}. By lemma 34, $L_{D_g} \in \text{SPACE}(g)$.
- Now, assume $g \not\leq_O f$. We prove that $L_{D_g} \notin \text{SPACE}(f)$.
- Assume for the sake of a contradiction that $L_{D_g} \in \text{SPACE}(f)$.
- By Lemma 30 there exists a Turing machine M and a k such that M decides L_D and

$$L_{D_g} = \{ \alpha \mid \mathcal{U}_f \text{ accepts } 0^a 1 0^b \$ \langle M \rangle \langle \alpha \rangle \} \qquad (*)$$

  for any $a, b \geq k$.
- By Lemma 36, there exist infinity many n such that $g(n) > af(n) + b$.
- Now, let σ be the string $0^a 1 0^b \$ \langle M \rangle$ for some a, b such that $a, b \geq k$ and $g(|\sigma|) > af(|\sigma|) + b$. We keep a and b fixed for the rest of this proof.
- **(Claim)**
  $\mathcal{U}_f$ accepts $0^a 1 0^b \$ \langle M \rangle \langle \sigma \rangle$ iff $\mathcal{U}_g$ accepts $0^1 1 0^0 \$ \langle M \rangle \langle \sigma \rangle$.

  We prove the claim.

  First we observe that the inputs is of the form $0^c 1 0^d \$ \ldots \langle \triangleright \rangle z \langle \sqcup \rangle$, so that $\mathcal{U}_f$ and $\mathcal{U}_g$ will not reject in the first stage of the definition of $\mathcal{U}$.

  - Assume that $\mathcal{U}_f$ accepts $0^a 1 0^b \$ \langle M \rangle \langle \sigma \rangle$.
  - The only differences between $\mathcal{U}_g$ on input $0^1 1 0^0 \$ \langle M \rangle \langle \sigma \rangle$ and $\mathcal{U}_f$ on input $0^a 1 0^b \$ \langle M \rangle \langle \sigma \rangle$, is that $\mathcal{U}_g$ marks off $g|\sigma|)$ cells for the simulation of M instead of $af(|\sigma|) + b$, and simulates for maximum $4^{g(|\sigma|)}$ steps instead of $4^{af(|\sigma|)+b}$.
  - Since $g(|\sigma|) > af(|\sigma|) + b$, $\mathcal{U}_g$ has more tape cells and number of steps to simulate M than $\mathcal{U}_f$.
  - Thus $\mathcal{U}_g$ accepts $0^1 1 0^0 \$ \langle M \rangle \langle \sigma \rangle$, and the proof of the implication from the left to the right is completed.
  - Assume that $\mathcal{U}_g$ accepts $0^1 1 0^0 \$ \langle M \rangle \langle \sigma \rangle$.
  - By definition of $\mathcal{U}_g$, M accepts σ.
  - Then $\sigma \in L_D$ since M decides L_D.
  - By (*), $\mathcal{U}_f$ accepts $0^a 1 0^b \$ \langle M \rangle \langle \sigma \rangle$, which proves the claim.
- Now we have

$$\begin{aligned}
\sigma \in L_{D_g} &\Leftrightarrow \mathcal{U}_f \text{ accepts } 0^a 1 0^b \$ \langle M \rangle \langle \sigma \rangle && (*) \\
&\Leftrightarrow \mathcal{U}_g \text{ accepts } 0^1 1 0^0 \$ \langle M \rangle \langle \sigma \rangle && \text{(Claim)} \\
&\Leftrightarrow D_g \text{ rejects } \sigma && \text{(by lemma 35)} \\
&\Leftrightarrow \sigma \notin L_{D_g} .
\end{aligned}$$

  This is a contradiction.

- Thus, we conclude that $L_{D_g} \notin \text{SPACE}(f)$.

□

The following definition is an attempt to make tighter space complexity classes. It only differs from how we defined our space complexity classes in the way that having a space bound $g \leq_O f$ is replaced by having a space bound f.

Definition 38 (Strict space complexity class) *Let $f\colon \mathbb{N} \to \mathbb{R}^+$. A **strict space complexity class** $STRICTSPACE(f)$ is the set of all languages decidable by a Turing machine with space bound f.*

The following theorem is the honest version of the well known tape compression theorem. We will give a detailed proof. As with the space hierarchy theorem, Hartmanis et al. formulated it in 1965 in [4], and gave a short proof. This theorem, and the following corollary, makes it clear why the Turing machine model is not suitable for separating space complexity classes that only differ by a constant factor.

Theorem 39 (Tape compression theorem) *Let f, g be honest functions such that $g \leq_O f$. Then,*

$$STRICTSPACE(g) \subseteq STRICTSPACE(f)$$

Proof.

- Let f, g be honest functions such that $g \leq_O f$, and assume that $L \in \text{STRICTSPACE}(g)$.
- Then there exist a Turing machine M_g deciding L with space bound g. Since $g \leq_O f$ there exists $c, n_0 \in \mathbb{N}$ such that $g(n) \leq cf(n)$ for all $n \geq n_0$. The above statement obviously holds for any larger n_0. Since g is monotone and space constructible, and therefore $log \leq_O g$, we let n_0 be such that $g(n) \geq 2c$ for all $n \geq n_0$.
- Let M_g have input tape alphabet Σ and work tape alphabet Γ, and let $\Lambda = \Gamma \cup \Gamma'$, where $\Gamma' = \{\boxed{\gamma} \mid \gamma \in \Gamma\}$. We draw a square around a symbol to denote the position of the work tape head of M_g.
- Let M_f be a Turing machine with work tape alphabet $\Delta = \Lambda \cup \Lambda \times \Lambda \cup \cdots \cup \Lambda^{2c}$. Notice that any chunk of symbols $\gamma_1 \ldots \gamma_i \in \Gamma^*$, for $i \leq 2c$, is represented by one symbol in Δ. This means that $g(|\alpha|)$ number of symbols on the work tape of M_g is compressed into $\frac{g(|\alpha|)}{2c} + 1$ number of symbols on the work tape of M_f.
- We are constructing M_f so that it behaves as M_g. On input α, M_f runs as follows:

  (1) Scan across the input tape from left to right entering a new state for each symbol read. If M_f reaches end of the input before entering the $n_0 + 1$ first states, it moves the input head to the left

end of the input tape, and proceeds with stage (2). Otherwise, $|\alpha| > n_0$. The head is moved to the leftmost symbol on the input tape and we proceed from stage (3).

(2) Compare α with every possible $s \in \Sigma^*$ such that $|s| \leq n_0$. Because there is a finite number of strings to check, M_f can remembering the current read symbols using the states only. This is done one by one (i.e. $\alpha = 0?$, $\alpha = 1?$, $\alpha = 01\ ?$, ...). M_f accepts if M_g accepts α, and rejects if M_g rejects α.

(3) For each state of M_g, each symbol on the input tape, and each symbol in Δ, we let M_f have a state that replace the symbol in the way that corresponds to what M_g writes, and how the work tape head of M_g moves. For instance, if the state q of M_g write 1 and move the work tape head rightwards when it reads 0 on both of the tapes, then, the corresponding state of M_f will replace the single symbol $00\boxed{0}0$ with $001\boxed{0}$ when 0 is read on the input tape. For each state of M_f that moves the square to the next chunk, we let M_f have an extra state which alters the chunk next to it in the expected way. M_f accepts if M_g accepts, and rejects if M_g rejects.

- For $|\alpha| \leq n_0$, M_f runs without moving the work tape head at all.
- If $|\alpha| > n_0$, then M_f decides L by scanning no more than $\frac{g(|\alpha|)}{2c} + 1$ cells. Since for all $n \geq n_0$ we have that $g(n) \geq 2c$ and $g(n) \leq cf(n)$, it follows that

$$\frac{g(|\alpha|)}{2c} + 1 \leq \frac{g(|\alpha|)}{2c} + \frac{g(|\alpha|)}{2c} = \frac{g(|\alpha|)}{c} \leq f(|\alpha|)$$

- Thus, M_f has space bound f, and $L \in \text{STRICTSPACE}(f)$ follows.

□

The following corollary states the relationship between the seemingly stricter space classes and the usual space classes.

Corollary 40 *Let f, g be honest functions.*

$$L \in \mathit{STRICTSPACE}(f) \Leftrightarrow L \in \mathit{SPACE}(f)$$

Proof.

- Let f, g be honest functions.
- For the right direction, assume that $L \in \text{STRICTSPACE}(f)$.
- Then there is a Turing machine M deciding L with space bound f.
- Since $\leq_O$ is reflexive (lemma 10), M has space bound $f \leq_O f$.
- Thus $L \in \text{SPACE}(f)$.
- For the left direction, assume that $L \in \text{SPACE}(f)$.

- Then there is a Turing machine M' deciding L with space bound $g \leq_O f$.
- Since f, g is honest, it follows from the previous theorem that $\text{STRICTSPACE}(g) \subseteq \text{STRICTSPACE}(f)$.
- Then $L \in \text{STRICTSPACE}(f)$.

□

The following lemma is a weaker version of theorem 39, which we will use in the next theorem.

Lemma 41

$$g \leq_O f \Rightarrow SPACE(g) \subseteq SPACE(f)$$

Proof.

- Let $g \leq_O f$, and assume that $L \in \text{SPACE}(g)$.
- Then there exists a Turing machine M deciding L with space bound $h \leq_O g$.
- Since $\leq_O$ is transitive (lemma 11), M decides L with space bound $h \leq_O f$.
- Thus, $L \in \text{SPACE}(f)$

□

In the next chapter, where we introduce a degree theory, it would be nice if we can abstract away the notion of space classes, and just work with the relation $\leq_O$. That is exactly what the following theorem enables us to do, which makes it one of our main goals in this thesis. It is also the reason for why we presented the stronger version of the space hierarchy theorem, since the weaker space hierarchy theorem will not be sufficient for the following to hold.

Theorem 42 *Let f and g be honest functions. Then*

$$g \leq_O f$$

$$\Leftrightarrow$$

$$SPACE(g) \subseteq SPACE(f)$$

$$\Leftrightarrow$$

$$STRICTSPACE(g) \subseteq STRICTSPACE(f)$$

Proof.

- Let f, g be honest functions.
- By lemma 41,

$$g \leq_O f \Rightarrow \text{SPACE}(g) \subseteq \text{SPACE}(f)$$

- Assume that $g \not\leq_O f$. Since f, g are honest, it follows from theorem 37 that we have a language L such that $L \in \text{SPACE}(g)$ and $L \notin \text{SPACE}(f)$, which means that $\text{SPACE}(g) \not\subseteq \text{SPACE}(f)$. Thus,

$$g \leq_O f \Leftarrow \text{SPACE}(g) \subseteq \text{SPACE}(f)$$

- Using that f, g are honest, we get from corollary 40 that,

$$\text{SPACE}(g) \subseteq \text{SPACE}(f) \Leftrightarrow \text{STRICTSPACE}(g) \subseteq \text{STRICTSPACE}(f)$$

□

Before we are ready to formulate the Space Hierarchy Theorem, as found in Sipser [3], we need the notion of little-o. Here is the definition as given by Sipser.

Definition 43 (Little-o) *Let $f, g \colon \mathbb{N} \to \mathbb{R}^+$ be functions. $g(n) = o(f(n))$ if*

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} \;=\; 0\,.$$

The following lemma relates little-o with our $\leq_O$, and will be useful during our proof of the space hierarchy theorem.

Lemma 44 *If $f(n) = o(g(n))$, then $g \not\leq_O f$.*

Proof.

(1) Let $f(n) = o(g(n))$.

(2) By definition 43

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \;=\; 0\,.$$

(3) Thus, for any $\epsilon > 0$ there exist n_0 such that for all $n > n_0$

$$\frac{f(n)}{g(n)} < \epsilon$$
$$\Leftrightarrow$$
$$g(n) > \frac{1}{\epsilon} f(n)$$

(4) Since we may let ϵ be arbitrary close to 0, $c = \frac{1}{\epsilon}$ may be arbitrary large. Thus, for any $c > 0$ there exist n_0 such that for all $n > n_0$

$$g(n) > cf(n)$$

(5) Then, there does not exist $c, n_0 \in \mathbb{N}$ such that $g(n) \leq cf(n)$ for all $n \geq n_0$. Thus, $g \not\leq_O f$. □

We will now state the space hierarchy theorem as found in [3], but for honest functions, and then show that it follows from our stronger space hierarchy theorem.

Theorem 45 (Space hierarchy theorem [3]) *For any honest function g there exists a language that is decidable in $O(g)$ space but not in $o(g)$ space.*

Proof.

- Let f be any honest function such that $f = o(g)$. It is sufficient to prove that there exists a language L that is decidable in $O(g)$ space but not in $O(f)$ space.
- By Lemma 44, we have $g \not\leq_O f$.
- By Theorem 37, we have a language L such that $L \in \text{SPACE}(g)$ and $L \notin \text{SPACE}(f)$.
- Hence, L is a language decidable in $O(g)$ space but not in $O(f)$ space.

□

In the proof of the following lemma we will see that there are functions separating space complexity classes with the stronger space hierarchy theorem (theorem 37), which can not be used for separating space classes with the space hierarchy theorem given above. The functions presented here are those we saw in the introduction, where we also saw a picture of their behaviour.

Theorem 46 *There exists honest functions $f, g\colon \mathbb{N} \to \mathbb{N}$ such that*

$$g \not\leq_O f \text{ and } f(n) \notin o(g(n))$$

Proof.

- Let $f, g\colon \mathbb{N} \to \mathbb{N}$ be defined as:

$$f(n) = \begin{cases} 2 & \text{if } n = 0 \\ f(n-1)^2 & \text{if n is even and } n > 0 \\ f(n-1)+1 & \text{if n is odd and } n > 0 \end{cases}$$

$$g(n) = \begin{cases} 2 & \text{if } n = 0 \\ g(n-1)+1 & \text{if n is even and } n > 0 \\ g(n-1)^2 & \text{if n is odd and } n > 0 \end{cases}$$

- The following shows that f and g is honest:

  (1) $f(n) \leq f(n+1)$ follows from the definition of f. Likewise for g.

  (2) $log_2(n+2) \leq f(n)$ holds by the following inductive proof.

  - For $n = 0$, $log_2(0+2) = 1 \leq 2 = f(0)$.
  - Assume that, for some k, $log_2(k+2) \leq f(k)$.
  - If k is even, $log_2((k+1)+2) \leq log_2(k+2)+1 \leq f(k)+1 = f(k+1)$.

– If k is odd, $log_2((k+1)+2) \leq log_2(k+2)+1 \leq f(k)+1 \leq f(k)^2 = f(k+1)$.

$log_2(n+2) \leq g(n)$ holds by the same proof, but replacing f with g and "odd" with "even".

(3) The following proves that $f(n)$ is space constructible:

– Let $\alpha = \alpha_1 \dots \alpha_n$. The following Turing machine M will compute $f(n)$ in binary.
– Write 10 on the work tape. If $\alpha = \epsilon$, the computation is finished and with $S_M deciding = 2$, and $2 \leq_O f$.
– Otherwise, write 1$ next to 10.

$$\llcorner 1\$10 \lrcorner$$

– We will refer to the numbers on the tape as

$$\llcorner k\$x_1 \lrcorner$$

All the variables and numbers on the tape will be in binary, possibly padded with zeros on the left.

– For each symbol of α, repeat the following: If $k = 1$, replace k by 0, and replace x_1 by $x_1 + 1$. Else if $k = 0$, replace k by 1. Copy x_1 two times, separated by $-symbols.

$$\llcorner k\$x_1\$x_1\$x_1 \lrcorner$$

Compute x_1^2 by adding x_1 to x_1, x_1 number of times.

$$\llcorner k\$x_1'\$x_2'\$x_1^2 \lrcorner$$

Shift x_1^2 to the left, overwriting $x_1'\$x_2'\$$.

$$\llcorner k\$x_1^2 \lrcorner$$

– When M has iterated this n times, replace $k$$ with $\llcorner\llcorner$.

$$\llcorner f(n) \lrcorner$$

– A space bound for this computation is

$$1+1+\lfloor log_2(f(n))+1 \rfloor + 1 + \lfloor log_2(f(n))+1 \rfloor + 1 + \lfloor log_2(f(n)^2)+1 \rfloor + 1$$

$$\leq 3log_2(f(n)^2) + 8 \leq 3f(n) + 8 \leq 7f(n)$$

Where the last inequality holds since $f(n) \geq 2$ for all n. Let $h(n) = 7f(n)$

Thus M has space bound $h \leq_O f$ such that on any input α of length n, M halts with $f(n)$ on the tape written in binary, which means that f is space constructible. The same holds for g by the same proof where we switch the case $k = 1$ with $k = 0$. **Thus f and g are honest.**

- The following proves that $g \not\leq_O f$:

  We have to prove that for any $c, n_0 \in \mathbb{N}$ there exists $n \geq n_0$ such that

  $$g(n) > cf(n)$$

  It is sufficient to prove that

  $$g(2c+1) > cf(2c+1) \tag{6.1}$$

  and for any $k \geq c$,

  $$g(2k+1) > cf(2k+1) \tag{6.2}$$

  We prove 6.1 with induction on c. First we check the base cases $c = 0, c = 1, c = 2$. Then we prove the inductive step for $c \geq 2$.

  $$g(2 \times 0 + 1) = g(1) = 2^2$$
  $$0 \times f(2 \times 0 + 1) = 0$$

  Thus $g(2 \times 0 + 1) > 0 \times f(2 \times 0 + 1)$.

  $$g(2 \times 1 + 1) = g(3) = g(2)^2 = (g(1) + 1)^2 = (2^2 + 1)^2 = 25$$
  $$1 \times f(2 \times 1 + 1) = f(3) = f(2) + 1 = f(1)^2 + 1 = (2 + 1)^2 + 1 = 10$$

  Thus $g(2 \times 1 + 1) > 1 \times f(2 \times 1 + 1)$.

  $$g(2 \times 2 + 1) = g(5) = g(4)^2 = (g(3) + 1)^2 = (25 + 1)^2 = 676$$
  $$2 \times f(2 \times 2 + 1) = 2 \times f(5) = 2(f(4) + 1) = 2(f(3)^2 + 1) = 2(10^2 + 1) = 202$$

  Thus $g(2 \times 2 + 1) > 2 \times f(2 \times 2 + 1)$.

  Assume that for some $k > 2$

  $$g(2k+1) > kf(2k+1) \tag{I.H.}$$

  Then it follows that

  $$\begin{aligned}
  g(2(k+1)+1) &= g(2k+3) = g(2k+2)^2 && \text{(definition of g)} \\
  &= (g(2k+1)+1)^2 && \text{(definition of g)} \\
  &= g(2k+1)^2 + 2g(2k+1) + 1 \\
  &> k^2 f(2k+1)^2 + 2kf(2k+1) + 1 && \text{(by (I.H.))} \\
  &> (k+1) f(2k+1)^2 + (k+1) && \text{(since } k > 2\text{)} \\
  &= (k+1)(f(2k+1)^2 + 1) \\
  &= (k+1)(f(2k+2)+1) && \text{(definition of f)} \\
  &= (k+1) f(2k+3) && \text{(definition of f)} \\
  &= (k+1) f(2(k+1)+1)
  \end{aligned}$$

Thus, equation (6.1) holds for any c. We prove that (6.2) holds for any $k \geq c$. Fix c. Observe that when $k = c$, (6.2) is (6.1). Now assume that for some $i \geq c$,

$$g(2i+1) > cf(2i+1) \qquad \text{(I.H.)}$$

Now we have that

$$\begin{aligned} g(2(i+1)+1) &= g(2i+1)^2 + 2g(2i+1) + 1 && \text{(definition of g)} \\ &> c^2 f(2i+1)^2 + 2cf(2i+1) + 1 && \text{(by (I.H.))} \\ &> c(f(2i+1)^2 + 1) \\ &= cf(2(i+1)+1) && \text{(definition of f)} \end{aligned}$$

Thus (6.2) holds for any $k \geq c$, and $g \not\leq_O f$. **Thus $\mathbf{g \not\leq_O f}$**

- To prove that $f(n) \notin o(g(n))$, we need to prove that

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} \quad \neq \quad 0\,.$$

which means that there exists $\epsilon > 0$ such that for all n_0 and some $n \geq n_0$,

$$f(n) \geq \epsilon g(n)$$

Let $\epsilon = 1$. It is sufficient to prove that $f(2k) \geq g(2k)$ for any k. We prove this by induction on k.

$$f(2 \times 0) = f(0) = 2 \geq 2 = g(0) = g(2 \times 0)$$

Assume that

$$f(2i) \geq g(2i) \qquad \text{(I.H.)}$$

for some i. Then

$$\begin{aligned} f(2(i+1)) = f(2i+1)^2 = (f(2i)+1)^2 &= f(2i)^2 + 2f(2i) + 1 \\ &\geq g(2i)^2 + 2g(2i) + 1 && \text{(I.H.)} \\ &> g(2i)^2 + 1 = g(2i+1) + 1 = g(2(i+1)) \end{aligned}$$

Thus $\mathbf{f(n) \notin o(g(n))}$

□

Chapter 7

The honest space degrees

In this chapter we present a neat degree theory for space complexity. This theory is the main motivation of this thesis.

Definition 47 ($\equiv_O$) *We define the relation $\equiv_O$ by*

$$f \equiv_O g \Leftrightarrow f \leq_O g \text{ and } g \leq_O f$$

Lemma 48 *$\equiv_O$ is an equivalence relation.*

Proof.

- By lemma 10, $\leq_O$ is reflexive. Thus $f \leq_O f$, which implies $f \equiv_O f$. Thus $\equiv_O$ is **reflexive**.
- Assume that $f \equiv_O g$ and $g \equiv_O h$. Then $f \leq_O g$, $g \leq_O f$, $g \leq_O h$, and $h \leq_O g$. By lemma 11, $\leq_O$ is transitive. Then, $f \leq_O h$, and $h \leq_O f$, which implies $f \equiv_O h$. Thus $\equiv_O$ is **transitive**.
- Assume $f \equiv_O g$. Then $f \leq_O g$ and $g \leq_O f$, which implies $g \equiv_O f$. Thus $\equiv_O$ is **symmetric**.
- Thus $\equiv_O$ is an **equivalence relation**.

□

We now introduce the honest space degrees.

Definition 49 ($\mathcal{H}$ of honest space degrees) *We let $\mathcal{H}$ denote the set of $\equiv_O$-equivalence classes of honest functions. The elements of $\mathcal{H}$ are the **honest space degrees**, or just **degrees**. We use boldface lowercase Latin letters $\mathbf{a}, \mathbf{b}, \mathbf{c}, \ldots$ to denote our degrees.*

Definition 50 ($\deg(f)$) *We let $deg(f)$ denote the degree of the honest function f, that is,*

$$deg(f) = \{g \mid g \equiv_O f\} .$$

We will need some basic relations similar to $\leq_O$ for comparing honest functions.

Definition 51 ($<_O$) *The relation $<_O$ is defined by*

$$f <_O g \Leftrightarrow f \leq_O g \text{ and } g \not\leq_O f$$

*We say that f **is strictly below** g and g **is strictly above** f if $f <_O g$.*

Not surprisingly, if f **lies below** g, or g **lies above** f, we will mean that $f \leq_O g$.

Definition 52 ($|_O$) *The relation $|_O$ is defined by*

$$f |_O g \Leftrightarrow f \not\leq_O g \text{ and } g \not\leq_O f .$$

We will use $<, \leq$, and $|$ to denote the relations induced on the degrees by $<_O, \leq_O, |_O$ respectively. E.g. if $\mathbf{a} = \deg(f)$ and $\mathbf{b} = \deg(g)$, we let $\mathbf{a} \leq \mathbf{b}$ mean that $f \leq_O g$. We say that $\mathbf{c}$ lies **strictly between** $\mathbf{a}$ and $\mathbf{b}$ if $\mathbf{a} < \mathbf{c} < \mathbf{b}$. We also say that $\mathbf{a}$ and $\mathbf{b}$ are **incomparable** if $\mathbf{a} \mid \mathbf{b}$.

If f and g are honest, we know that

$$f \leq_O g$$
$$\Leftrightarrow$$
$$\text{SPACE}(f) \subseteq \text{SPACE}(g)$$
$$\Leftrightarrow$$
$$\text{STRICTSPACE}(f) \subseteq \text{STRICTSPACE}(g)$$

from theorem 42. Thus, for any honest degree $\mathbf{a}$, we have

$$\text{SPACE}(f) = \text{SPACE}(g) \quad \text{for } f, g \in \mathbf{a}.$$

Furthermore, we have

$$\deg(f) \leq \deg(g) \quad \Leftrightarrow \quad \text{SPACE}(f) \subseteq \text{SPACE}(g) .$$

Definition 53 (Least upper bound) *The **least upper bound** of the degrees $\mathbf{a}$ and $\mathbf{b}$ is the degree $\mathbf{c}$ such that*

$$\mathbf{a} \leq \mathbf{c} \text{ and } \mathbf{b} \leq \mathbf{c}$$

and

$$\text{if } \mathbf{a} \leq \mathbf{c}_0 \text{ and } \mathbf{b} \leq \mathbf{c}_0 \text{ , then } \mathbf{c} \leq \mathbf{c}_0$$

This means that $\mathbf{c}$ lies above both $\mathbf{a}$ and $\mathbf{b}$, and below any other degree that lies above both $\mathbf{a}$ and $\mathbf{b}$

Definition 54 (Greatest lower bound) *The **greatest lower bound** of the degrees $\mathbf{a}$ and $\mathbf{b}$ is the degree $\mathbf{c}$ such that*

$$\mathbf{a} \geq \mathbf{c} \text{ and } \mathbf{b} \geq \mathbf{c}$$

and

$$\text{if } \mathbf{a} \geq \mathbf{c}_0 \text{ and } \mathbf{b} \geq \mathbf{c}_0 \text{ , then } \mathbf{c} \geq \mathbf{c}_0$$

This means that $\mathbf{c}$ lies below both $\mathbf{a}$ and $\mathbf{b}$, and above any other degree that lies below both $\mathbf{a}$ and $\mathbf{b}$.

Definition 55 (Lattice) *A degree structure where each pair of elements has both a least upper bound and a greatest lower bound is called a* ***lattice****.*

The notion of a lattice is applicable to many other structures as well, as the following example indicate.

Example 56 *Let L be the structure*

$$L = (\{K \mid K \subseteq \mathbb{N}\}, \subseteq)$$

Then, we let the ***least upper bound*** *on the sets A and B be the set C such that*

$$A \subseteq C \text{ and } B \subseteq C$$

and

$$\text{if } A \subseteq C_0 \text{ and } B \subseteq C_0 \text{ , then } C \subseteq C_0$$

Then, we let the ***greatest lower bound*** *on the sets A and B be the set C' such that*

$$A \supseteq C' \text{ and } B \supseteq C'$$

and

$$\text{if } A \supseteq C_0 \text{ and } B \supseteq C_0 \text{ , then } C' \supseteq C_0$$

L is a lattice, since, for any pair of sets A, B such that $A \subseteq \mathbb{N}$ and $B \subseteq \mathbb{N}$, the above statements holds for $C = A \cup B$ and $C' = A \cap B$.

Definition 57 (Join) *We define the* ***join*** *of the honest functions f and g, written* $\max[f, g]$*, by*

$$\max[f, g](x) \;=\; \max(f(x), g(x)) \; .$$

Definition 58 (Meet) *We define the* ***meet*** *of the honest functions f and g, written* $\min[f, g]$*, by*

$$\min[f, g](x) \;=\; \min(f(x), g(x)) \; .$$

The following lemma will be useful in the proof of the next lemma where we prove that $\min[f, g]$ is space constructible. We will there need a seemingly stronger version what it means to be space constructible. It will be useful for us to obtain an Turing machine M which use exactly $f(n)$ space for any large enough n, and not just $g(n)$ such that $g \leq_O f$.

Lemma 59 *Let f be an honest function. Then, there exists a positive integer n_0 and a Turing machine M, such that on any input α of length n, M halts with $f(n)$ on the tape written in binary, and $S_M(\alpha) = f(n)$ for all $n \geq n_0$.*

Proof.

- Let f be an honest function.

- Then f is space constructible. Let therefore M be a Turing machine with space bound $g \leq_O f$ such that on any input α of length n, M halts with $f(n)$ on the tape written in binary. By lemma 17, we let g be honest as well. Since $g \leq_O f$, there exists $c, k \in \mathbb{N}$ such that $g(n) \leq cf(n)$ for all $n \geq k$.

- Since f and g are honest, we can use the same technique as we saw in the proof of theorem 39. Then, for large enough work tape alphabet, M' can compute the same as M, but with compressed symbols. I.e. The single symbol 1010 on the work tape of M' corresponds to the four symbols 1010 on the work tape of M. By the same arguments as in the proof of theorem 39, for large enough $n_0 \geq k$, we have that $S_{M'}(\alpha) \leq f(n)$ for all $n > n_0$. When M' has computed $f(n)$ in a compressed way, it transforms each symbol on the work tape back to the corresponding symbols 0 and 1. This takes no more space than to write $f(n)$ in binary. To make M' use at least $f(n)$ space, we use that f is honest, and observe the above still holds if we continue to increase n_0. Therefore we let n_0 be large enough for M' to make a copy of $f(n)$ without using more than $f(n)$ space in total ($f(n) \geq 16$ should hold). This copy will be used as a counter. Then, M' decrements the counter while moving a \$ symbol one square away each time the counter does not decrease in length. When the counter becomes zero, M' erases the \$ symbol and the counter.

- Thus, there exists a positive integer n_0 and a Turing machine M', such that on any input α of length n, M' halts with $f(n)$ on the tape written in binary, and $S_{M'}(\alpha) = f(n)$ for all $n \geq n_0$.

□

We will now see several properties of the meat and join function.

Lemma 60 *Let f and g be honest functions. Then,* $\max[f,g]$ *and* $\min[f,g]$ *are honest functions.*

Proof.

(1) For (2) and (3) below, we prove the statements for n such that $\max[f,g](n) = f(n)$. The case where n is such that $\max[f,g](n) = g(n)$ holds by the same arguments when we replace f with g.

(2)

$$\max[f,g](n) = f(n) \leq f(n+1) \leq max(f(n+1), g(n+1))$$
$$\min[f,g](n) = g(n) \leq g(n+1) \leq \min[f,g](n+1)$$

(3)

$$\log_2(n+2) \leq f(n) = \max[f,g](n)$$

$$\log_2(n+2) \leq g(n) = \min[f,g](n)$$

(4) Let α be any input of length n, and M be a Turing machine with space bound $h \leq_O f$, such that M halts with $f(n)$ on the tape written in binary, and let M' be a Turing machine with space bound $h' \leq_O g$ such that M' halts with $g(n)$ on the tape written in binary. By lemma 17, we may obtain honest h and h' such that the above still holds. Let MAX be a Turing machine that first computes $f(n)$ by executing M, and then computes $g(n)$ by executing M'. Then MAX keeps the largest value. The following defines a space bound $t(n)$ for MAX such that $t \leq_O \max[f, g]$:

$$
\begin{aligned}
& h(n) + h'(n) + 1 \\
& \leq 3max(h(n), h'(n)) && \text{Honest implies } h(n), h'(n) \geq 1 \\
& \leq 3\max(cf(n), cg(n)) && h \leq_O f \text{ and } h' \leq_O g \\
& = 3c\max(f(n), g(n)) \\
& = t(n) && \text{defines t(n)}
\end{aligned}
$$

Thus $\max[f,g](n)$ space constructible and honest. Since f, g is honest, it follows from lemma 59 that there exists a positive integers n_0, n_0' and Turing machines N and N' such that for all $n \geq max(n_0, n_0')$:

- N halts with $f(n)$ on the tape written in binary
- $S_N(\alpha) = f(n)$.
- N' that halts with $g(n)$ on the tape written in binary
- $S_{N'}(\alpha) = g(n)$.

For the finite number of $n < max(n_0, n_0')$, we let MIN use the states only to count the length of n, and then write $\min[f, g](n)$ on the tape. Thus, MIN require less than $\min[f, g](n)$ space for this case, and $\min[f, g](n) \leq_O \min[f, g](n)$ by lemma 10. For $n \geq max(n_0, n_0')$, we let MIN be a Turing machine that first mark off two segments on the tape. Then, MIN executes one step of N with the work tape head of MIN inside the first segment. Then it moves to the second segment and executes one step of N'. Every time MIN needs to write a symbol where it is a marker, it makes the current segment one cell larger by shifting the outermost marker, and every symbol on the way back to the first marker. Then it makes the other segment one cell larger by shifting the outermost marker of that segment. Lets call the first out of N and N' that finishes for H, and the other for J. MIN then continues to execute J, but without shifting the markers. If J requires more space than H, then the value of H is the one to keep. Otherwise, MIN compares the values from H and J, and keep the minimum of them. MIN requires $\min[f, g](n)$ number of cells for each of N and N', plus a small finite number k of marking symbols. The following

defines a space bound $r(n)$ for MIN such that $r \leq_O \min[f,g]$:

$$\begin{aligned} & 2\min[f,g](n) + k \\ & \leq (2+k)\min[f,g](n) && \text{Honest implies } f(n), g(n) \geq 1 \\ & = r(n) && \text{defines r(n)} \end{aligned}$$

Thus $\min[f,g](n)$ space constructible and honest.

□

Lemma 61 *Let f, g, h be honest functions.*

(i) $\min[f,g] \leq_O f$ *and* $\min[f,g] \leq_O g$.

(ii) If $h \leq_O f$ *and* $h \leq_O g$, *then* $h \leq_O \min[f,g]$.

Proof.

(i) For all n, we have that $\min[f,g](n) \leq f(n)$.

- Let $c = 1$. Then $\min[f,g](n) \leq cf(n)$ for all $n \geq 0$.
- Thus $\min[f,g] \leq_O f$.
- The other case where $\min[f,g] \leq_O g$ follows by replacinging f with g in the argument above.

(ii) Assume that $h \leq_O f$ and $h \leq_O g$.

- Then there exist c_1, c_2, and n_0 such that for all $n \geq n_0$

$$h(n) \leq c_1 f(n)$$

$$h(n) \leq c_2 g(n)$$

- Then, for all $n \geq n_0$ such that $\min[f,g](n) = f(n)$, we have that

$$h(n) \leq c_1 f(n) \leq max(c_1, c_2)\min[f,g](n)$$

For all $n \geq n_0$ such that $\min[f,g](n) = g(n)$,

$$h(n) \leq c_2 g(n) \leq max(c_1, c_2)\min[f,g](n)$$

- Thus $h \leq_O \min[f,g]$.

□

Lemma 62 *Let f, g, h be honest functions.*

(i) $f \leq_O \max[f,g]$ *and* $g \leq_O \max[f,g]$.

(ii) If $f \leq_O h$ *and* $g \leq_O h$, *then* $\max[f,g] \leq_O h$.

Proof.

(i) For all n, we have that $f(n) \leq \max[f,g](n)$.

- Let $c = 1$. Then $f(n) \leq c \max[f,g](n)$ for all $n \geq 0$.
- Thus $f \leq_O \max[f,g]$.
- The other case where $g \leq_O \max[f,g]$ follows by replacinging f with g in the argument above.

(ii) Assume that $f \leq_O h$ and $g \leq_O h$.

- Then there exist c_1, c_2, and n_0 such that for all $n \geq n_0$

$$f(n) \leq c_1 h(n)$$

$$g(n) \leq c_2 h(n)$$

- Then, for all $n \geq n_0$ such that $\max[f,g](n) = f(n)$, we have that

$$\max[f,g](n) = f(n) \leq max(c_1, c_2) h(n)$$

For all $n \geq n_0$ such that $\max[f,g](n) = g(n)$,

$$\max[f,g](n) = g(n) \leq max(c_1, c_2) h(n)$$

- Thus $\max[f,g] \leq_O h$.

□

Lemma 63 *For any honest functions f, f_1, g, g_1 such that $f \leq_O f_1$ and $g \leq_O g_1$, we have that*

(i) $\min[f,g] \leq_O \min[f_1, g_1]$

(ii) $\max[f,g] \leq_O \max[f_1, g_1]$

Proof.

- Assume that f, f_1, g, g_1 are honest, and that $f \leq_O f_1$ and $g \leq_O g_1$.
- By (i) from lemma 61, it follows that $\min[f,g] \leq_O f$ and $\min[f,g] \leq_O g$.
- By lemma 11, $\leq_O$ is transitive. Then we have that $\min[f,g] \leq_O f_1$ and $\min[f,g] \leq_O g_1$.
- By (ii) from lemma 61, $\min[f,g] \leq_O \min[f_1, g_1]$.
- Thus, (i) holds.
- We can see that (ii) follows by the same argument if we replace $\max[f,g]$ with $\min[f,g]$, and use lemma 62 instead of lemma 61.

□

Notice that our previous lemma entails that

$$(f \equiv_O f_1 \text{ and } g \equiv_O g_1) \Rightarrow (\max[f,g] \equiv_O \max[f_1, g_1] \text{ and } \min[f,g] \equiv_O \min[f_1, g_1])$$

when f, f_1, g, g_1 are honest functions. By Lemma 60, we know that $\max[f,g]$ and $\min[f,g]$ are honest functions whenever f and g are. This justifies the following definition.

Definition 64 (The join $\cup$ and the meet $\cap$ of degrees) *Let f and g be honest functions such that $\deg(f) = \mathbf{a}$ and $\deg(g) = \mathbf{b}$. We define the* join of $\mathbf{a}$ and $\mathbf{b}$, *written $\mathbf{a} \cup \mathbf{b}$, by $\mathbf{a} \cup \mathbf{b} = deg(\max[f, g])$. We define the* meet of $\mathbf{a}$ and $\mathbf{b}$, *written $\mathbf{a} \cap \mathbf{b}$, by $\mathbf{a} \cap \mathbf{b} = deg(\min[f, g])$.*

As the following theorem states, our degree theory have the nice property of being a distributive lattice.

Theorem 65 (Distributive Lattice) *The structure $\langle \mathcal{H}, \leq, \cup, \cap \rangle$ is a distributive lattice, that is, for any $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathcal{H}$, we have that*

(i) $\mathbf{a} \cap \mathbf{b}$ *is the greatest lower bound of* $\mathbf{a}$ *and* $\mathbf{b}$ *under the ordering* $\leq$

(ii) $\mathbf{a} \cup \mathbf{b}$ *is the least upper bound of* $\mathbf{a}$ *and* $\mathbf{b}$ *under the ordering* $\leq$

(iii) $\mathbf{a} \cup (\mathbf{b} \cap \mathbf{c}) = (\mathbf{a} \cup \mathbf{b}) \cap (\mathbf{a} \cup \mathbf{c})$ *and* $\mathbf{a} \cap (\mathbf{b} \cup \mathbf{c}) = (\mathbf{a} \cap \mathbf{b}) \cup (\mathbf{a} \cap \mathbf{c})$

Proof.

- Let f, g, h, and i be honest functions such that $\deg(f) = \mathbf{a}$, $\deg(g) = \mathbf{b}$, $\deg(h) = \mathbf{c}_0$, and $\deg(i) = \mathbf{c}$.
- By definition 64, we have that $\mathbf{a} \cap \mathbf{b} = \deg(\min[f, g])$.
- By lemma 61 (i) we have that $f \geq_O \min[f, g]$ and $g \geq_O \min[f, g]$, which means that $\mathbf{a} \geq \mathbf{a} \cap \mathbf{b}$ and $\mathbf{b} \geq \mathbf{a} \cap \mathbf{b}$.
- By lemma 61 (ii) we have that if $f \geq_O h$ and $g \geq_O h$, then $\min[f, g] \geq_O h$, which means that if $\mathbf{a} \geq \mathbf{c}_0$ and $\mathbf{b} \geq \mathbf{c}_0$, then $\mathbf{a} \cap \mathbf{b} \geq \mathbf{c}_0$.
- Thus $\mathbf{a} \cap \mathbf{b}$ is the greatest lower bound of $\mathbf{a}$ and $\mathbf{b}$.
- The proof of (ii) is symmetric, where we use lemma 62 in place of lemma 61.
- (iii) holds since

$$\mathbf{a} \cup (\mathbf{b} \cap \mathbf{c}) = \deg(\max[f, \min[g, i]]) =$$
$$\deg(\min[\max[f, g], \max[f, i]]) = (\mathbf{a} \cup \mathbf{b}) \cap (\mathbf{a} \cup \mathbf{c})$$

and

$$\mathbf{a} \cap (\mathbf{b} \cup \mathbf{c}) = \deg(\min[f, \max[g, i]]) =$$
$$\deg(\max[\min[f, g], \min[f, i]]) = (\mathbf{a} \cap \mathbf{b}) \cup (\mathbf{a} \cap \mathbf{c})$$

□

We finish with the following conjecture.

Conjecture 66 (Density) *For any degrees $\mathbf{a}$ and $\mathbf{b}$, such that $\mathbf{a} < \mathbf{b}$, there exist degrees $\mathbf{c_1}$ and $\mathbf{c_2}$ such that*

(i) $\mathbf{a} < \mathbf{c_1} < \mathbf{b}$

(ii) $\mathbf{a} < \mathbf{c_2} < \mathbf{b}$

(iii) $\mathbf{a} = \mathbf{c_1} \cap \mathbf{c_2}$

(iv) $\mathbf{b} = \mathbf{c_1} \cup \mathbf{c_2}$

Appendices

Appendix A

More details of $\mathcal{U}$

In chapter 3 we defined the universal Turing machine $\mathcal{U}$ without giving all the necessary details. This chapter is intended to fill in these holes. Let's say $\langle c \rangle = [110]$ and $\langle c' \rangle = [11011]$ should replace one another, or the binary number $[11]$ is incremented, or $[100]$ decremented. Sometimes $\mathcal{U}$ needs more space in the middle of the tape. Other times $\mathcal{U}$ need to erase symbols and fill the erased space. $\mathcal{U}$ should manage these cases well.

We will here cover how $\mathcal{U}$ deals with

- replacing a symbol c with c' when $|\langle c' \rangle| > |\langle c \rangle|$
- incrementing an odd binary number
- replacing a symbol c with c' when $|\langle c' \rangle| < |\langle c \rangle|$
- decrementing a even binary number

Here is how $\mathcal{U}$ solves them.

We **need more space** when $\mathcal{U}$ is about to write $s \in \{0, 1\}$ at a cell containing]

$$\ldots s_1][\ldots$$

$\mathcal{U}$ will replace] by $*$.

$$\ldots s_1 * [\ldots$$

Move to the rightmost non-blank symbol of the work tape.

$$\ldots]_{\sqcup}\ldots$$

Move each symbol one cell to the right until it is back at $*$ again.

$$\ldots s_1 * [[\ldots$$

$*$ is replaced by s, and] is written at the new available cell to the right. Move one cell left.

$$\ldots s_1 s][\ldots$$

We might also need more space when **incrementing a binary number**, say [11].

$$\ldots [11]\ldots$$

$\mathcal{U}$ will replace [by $*$.

$$\ldots * 11] \ldots$$

Move to the rightmost non-blank symbol of the work tape.

$$\ldots] \sqcup \ldots$$

Move each symbol one cell to the right until it's back at $*$ again.

$$\ldots * 111] \ldots$$

$*$ is replaced by [. Then, make two steps to the right, and write 0's until].

$$\ldots [100] \ldots$$

Move left until [

$$\ldots [100] \ldots$$

We **need to erase** some cells when $\mathcal{U}$ are about to write] at a cell containing 0 or 1.

$$\ldots [s_1 s_2 s_3] [\ldots$$

$\mathcal{U}$ will write].

$$\ldots [s_1] s_3] [\ldots$$

Move right while writing $*$, until the first], which is also replaced.

$$\ldots [s_1] * *[\ldots$$

Move one step to the right.

$$\ldots [s_1] * *[\ldots$$

Swap it with the $*$ to the left.

$$\ldots [s_1] * [* \ldots$$

This swapping continues rightwards until $\sqcup$. Then, $*$ is replaced by $\sqcup$.

$$\ldots [s_1] * [\ldots] \sqcup$$

Move left until $*$ and continue the swapping. If there are none $*$'s left, $\mathcal{U}$ moves to the leftmost non-blank symbol.

$$\sqcup [\ldots [s_1] [\ldots$$

We may also need to erase some cells when **decrementing a binary number**, say [100].

$$\ldots [100] [\ldots$$

Move rightwards, replacing every 0 with 1 until].

$$\ldots [111] [\ldots$$

Move left and write].

$$\ldots [11]][\ldots$$

Move two steps to the right.

$$\ldots [11]][\ldots$$

Swap with the symbol to the left.

$$\ldots [11][]\ldots$$

Move one step right, and continue to swap] rightwards until ␣ .

$$\ldots]]$$

Write ␣ .

$$\ldots]␣$$

Recall the form of the left end of the tape.

$$␣\langle q[b]\ldots\rangle$$

To move back at the binary number again, move the head left until ␣ , then right past the first].

$$␣\langle q[11]\ldots\rangle$$

Appendix B

Detailed description of $\mathcal{U}_f$

In chapter 4 we defined the space bound universal Turing machine $\mathcal{U}_f$ without giving much details on how it works. But, since we like details, the following is intended to satisfy us. Each time we use the terms x_1, x_2, $f(|\alpha|)$, $af(|\alpha|)$ or $af(|\alpha|)+b$ in this description, the tape will actually contain the terms written as binary numbers. We expect the input β to be of the form

$$\beta = 0^a 1 0^b \$ \langle M \rangle \langle \alpha \rangle$$

Stage 1.

First stage is to verify that the input β is of the form

$$0^a 1 0^b \$ \ldots \langle \triangleright \rangle z \langle \llcorner \rangle$$

for any $a, b \geq 0$ and $z = s_1 \ldots s_n$, where s_i is of the form $[x]$ such that $x \in \{0,1\}^*$ for $0 \leq i \leq n$. Starting at the leftmost non-blank symbol, read zero or more 0 symbols, then a 1 symbol, then zero or more 0 symbols, and then the symbol \$. Then continue rightwards until reading $[1]$, which is $\langle \triangleright \rangle$. Then read zero or more strings on the form $[s]$, where $s \in 0, 1^*$. Each time $[10]$ is read, go to stage 2. if the next symbol is $\llcorner$. Otherwise, reject if $\llcorner$ is read after something different from $[10]$. If any of these searches fail, then reject.

Stage 2.

Since f is honest, and therefore space constructible, there exits a Turing machine N with space bound $h \leq_O f$, such that on any input of length $|\lambda|$, N halts with $f(|\lambda|)$ on the tape written in binary. $\mathcal{U}_f$ will run as N does, but it will treat $\langle \triangleright \rangle$ and $\langle \llcorner \rangle$ as N treats $\triangleright$ and $\llcorner$ respectively. Any sequence $[x]$, where $x \in \{0,1\}^*$, will be treated as N treats one symbol of α. With $f(|\alpha|)$ on the work tape, $\mathcal{U}_f$ can compute $af(|\alpha|)+b$ in the following way.

1. Make a copy x_1 of $f(|\alpha|)$, so that the work tape is of the form

$$0 \$ x_1 \$ f(|\alpha|)$$

2. For each symbol of 0^a, set $x_1 = f(|\alpha|)$ and add x_1 to the leftmost number.

$$af(|\alpha|)\$x_1\$f(|\alpha|)$$

3. Then, increment $af(|\alpha|)$ for each symbol of 0^b.

$$af(|\alpha|) + b \ \ \$x_1\$f(|\alpha|)$$

4. Erase $x_1\$f(|\alpha|)$ and write a \$-sign to the left of $af(|\alpha|) + b$

$$\$af(|\alpha|) + b\$$$

5. Let $w = af(|\alpha|) + b$ so that we can denote the current work tape as $\$w\$$. Subtract 1 from w and move the rightmost \$-sign one square to the right if w and $w - 1$ has the same number of digits. Continue until $w = 0$, and then replace 0 with $\sqcup$.

$$\$ \underbrace{\sqcup \cdots \sqcup}_{af(|\alpha|)+b} \$$$

Stage 3.

Now, $\mathcal{U}_f$ computes $4^{af(|\alpha|)+b}$. Observe that $4^{af(|\alpha|)+b}$ written in binary is $1 \underbrace{0 \ldots 0}_{2(af(|\alpha|)+b)}$. Write 00 for each of the $af(|\alpha|) + b$-blanks between the two \$-symbols on the work tape, and then a 1 to the left of them.

$$1 \underbrace{0 \ldots 0}_{2(af(|\alpha|)+b)} \$ \underbrace{\sqcup \cdots \sqcup}_{af(|\alpha|)+b} \$$$

Stage 4.

Now, $\mathcal{U}_f$ moves the input head to q_0, and the work head to the $\sqcup$ on the left side of the rightmost \$ symbol, and is ready to simulate M.

$$\triangleright 0^a 1 0^b \$ \langle q_0 \delta \rangle \langle \triangleright \alpha_1 \ldots \alpha_n \sqcup \rangle \sqcup$$

$$x_2 \$ \underbrace{\sqcup \cdots \sqcup}_{af(|\alpha|)+b} \$$$

where $x_2 = 4^{af(|\alpha|)+b}$. During this simulation, $\mathcal{U}_f$ will execute an altered version $\mathcal{U}_{lim}$ of $\mathcal{U}$. We construct $\mathcal{U}_{lim}$ by first letting it be the same as $\mathcal{U}$. Then, $\mathcal{U}_{lim}$ will treat any \$-symbol at the input tape as $\mathcal{U}$ treats $\sqcup$. Let any state of $\mathcal{U}_{lim}$ reject if a \$-sign is read from the work tape (we will soon add states processing \$-signs without halting). Then, let any transition rule that moves the work head to the right go to an intermediate subroutine where $\mathcal{U}_{lim}$ checks the symbol on the work tape. If it is not \$, $\mathcal{U}_{lim}$ continues as $\mathcal{U}$. If it is \$, $\mathcal{U}_{lim}$ needs to shift every symbol in the marked area on the work

taped one square to the left. This shifting is done by first moving leftwards until reading a $\sqcup$. If a \$-sign is read before a $\sqcup$, $\mathcal{U}_{lim}$ rejects. Otherwise, it shifts every symbol which is to the right of the $\sqcup$-symbol one cell to the left. The shifting stops when it reads \$ again, then it moves one step left, writes a $\sqcup$, and continues as $\mathcal{U}$.

Each time $\mathcal{U}_{lim}$ is on the 8'th step of the *Simulation stage*, and before going to the *Halting check*, go left until \$, and then one more step to the left so that the work tape head is over x_2. Decrement x_2. If $x_2 = 0$, reject, otherwise, go rightwards until \$, and then rightwards until the first non-blank symbol after \$. Then, go to the *Halting check*.

Stage 5.

$\mathcal{U}_f$ will accept if $\mathcal{U}_{lim}$ accepts, and reject if $\mathcal{U}_{lim}$ rejects.

Appendix C

Detailed description of D_g

In the last chapter we made the definition of $\mathcal{U}_f$ more precise. Here we will do the same for the Turing machine D_g, which were given in chapter 4 in definition 31.

We will construct D_g so that when the input tape is

$$\triangleright 0^a 1 0^b \$ \langle M \rangle \llcorner$$

it does the opposite of what $\mathcal{U}_g$ does when the input tape is

$$\triangleright 0^1 1 0^0 \$ \langle M \rangle \langle \triangleright 0^a 1 0^b \$ \langle M \rangle \llcorner \rangle \llcorner$$

where 0^a, 0^b, and $\langle M \rangle$ is written out before we apply $\langle\rangle$ on $\triangleright 0^a 1 0^b \$ \langle M \rangle \llcorner$. Let $\sigma = 0^a 1 0^b \$ \langle M \rangle$. When encoding σ as $\langle \sigma \rangle$, each symbol in $\{0, 1, \$, [,]\}$ is assigned an associated number $i \in \{5, \ldots, |\{0, 1, \$, [,]\}| + 4\}$ If b equal i written in binary,

$$\langle c_i \rangle \equiv [b]$$

Since we may freely choose which symbol gets associated with which number, we make a choice.

$$\langle 0 \rangle \equiv [101]$$
$$\langle 1 \rangle \equiv [110]$$
$$\langle \$ \rangle \equiv [111]$$
$$\langle [\rangle \equiv [1000]$$
$$\langle] \rangle \equiv [1001]$$

D_g runs as follows:

Stage 1.

Notice in the definition of D_g, g is an honest function and therefore space constructible. Thus, computing $g(|\sigma|)$ in binary can be done by a Turing machine M having space bound $h \leq_O g$. Let D_g first run M, then write a \$-symbol at each side of $g(|\sigma|)$.

$$\$ g(|\sigma|) \$$$

Then, let $w = g(|\sigma|)$ so that we can denote the current work tape as $\$w\$$. Subtract 1 from w and move the rightmost \$-sign one square to the right if w and $w-1$ has the same number of digits. Continue until $w = 0$, and then replace 0 with $\llcorner$.

$$\$\underbrace{\llcorner \cdots \llcorner}_{g(|\sigma|)}\$$$

Stage 2.

Now, D_g computes $4^{g(|\sigma|)}$. Observe that $4^{g(|\sigma|)}$ written in binary is $1\underbrace{0\ldots0}_{2g(|\sigma|)}$. Write 00 for each of the $g(|\sigma|)$-blanks between the two \$-symbols on the work tape, and then a 1 to the left of them.

$$1\underbrace{0\ldots0}_{2g(|\sigma|)}\$\underbrace{\llcorner \cdots \llcorner}_{g(|\sigma|)}\$$$

The Turing machine $\mathcal{U}_g$ where given in detail in the last chapter. Let D_g simulate M on σ by executing an altered version of what is explained in step 4. of $\mathcal{U}_g$. If the counter for each step of M ever gets zero, accept, which is the opposite of what $\mathcal{U}_g$ would have done. Let each action done on the work tape be as $\mathcal{U}_g$, unless otherwise is stated.

- To move the input head to q_0 in $\langle M \rangle$, go left until $\triangleright$, then right until $[1001]$, which is $\langle \$ \rangle$, and then one step to the right.
- $\mathcal{U}_{lim}$ treat any \$-sign at the input tape as $\mathcal{U}$ treat $\llcorner$. Use the original rules of $\mathcal{U}$ here, instead of replacing them by rules about \$-sign at the input tape.
- When $\mathcal{U}_{lim}$ executes $\mathcal{U}$, at the initialize work tape stage, step 2, when moving the input head leftwards until $\langle \triangleright \rangle$, continue until $\triangleright$ instead. Also count every symbol, including the $\llcorner$-sign, and not just $]$.
- At the initialize work tape stage, step 3, copy α_1, which in our case is $\langle 0 \rangle$, to the work tape. This is done by moving one step right on the input tape, and then writing $[101]$ on the work tape.
- At the simulation stage of $\mathcal{U}$, step 7., when $\mathcal{U}$ moves input head leftwards while decrementing π for each $[$, start decrementing at $\llcorner$, and decrement for each symbol, not just $[$, until $\pi = 0$. Then move one step left instead of until the next $[$. When replacing c by x, use the encoding given above. I.e. replace c by $[101]$ is 0 is read from the input tape. When $\mathcal{U}$ moves the input head rightwards until the symbol $\llcorner$while incrementing π for each $[$, increment for each symbol read instead, including $\llcorner$. The right case is done similarly.

Stage 3.

If the simulation requires more space than marked, *accept*.

Stage 4.

If M accepts, *reject*. If M rejects, *accept*.

Bibliography

[1] Viliam Geffert. 'Space hierarchy theorem revised'. In: *Theoretical Computer Science* 295.1–3 (2003), pp. 171–187. ISSN: 0304-3975.

[2] Harry R. Lewis and Christos H. Papadimitriou. *Elements of the Theory of Computation*. 2nd. Prentice Hall PTR, 1997. ISBN: 0132624788.

[3] Michael Sipser. *Introduction to the Theory of Computation*. Course Technology CENGAGE Learning, 2006. ISBN: 9780619217648.

[4] R.E. Stearns, J. Hartmanis and P.M. Lewis. 'Hierarchies of memory limited computations'. In: *Switching Circuit Theory and Logical Design, 1965. SWCT 1965. Sixth Annual Symposium on*. Oct. 1965, pp. 179–190.

[5] Alan Mathison Turing. 'On computable numbers, with an application to the Entscheidungsproblem'. In: *J. of Math* 58.345-363 (1936), p. 5.