In the theory of computing devices and computation, many different models have been distinguished, such as Turing machines, Boolean circuits, finite automata and neural networks. These models can divided into two classes: models that only take inputs of a fixed length (i.e., with a possibly different device or program for every input length) and models that take inputs of any arbitrary length. Both classes lead to a rich complexity theory (cf. Balcázar et al.[1]). We are especially interested in the broad class of models of the former type.

We can model devices in this class by considering sequences of machines, one machine for each input length. Sequences of Boolean circuits are a prime instance of this class, but so are the various types of machines using advice (i.e., with a possibly different advice value for every input length). Since the sequences can be arbitrary, there is in general no way to generate a sequence recursively, that is, we cannot give a uniform description of the machines of the sequence. Therefore, we call these sequences of computing devices non-uniform models and the theory based on these models non-uniform complexity theory.

The concept of advice functions, which was introduced by Karp and Lipton[22], is thoroughly connected with sequences. We make heavy use of sequences throughout this thesis, and advice functions prove to be a useful tool. In particular, we are interested in the way different advice functions can lead to different complexities of computation, when added to familiar machine models.

It is known that for every pair of exponentially bounded, integer-valued functions $f$ and $g$, there are languages that can be decided by a Turing machine with an advice of size $g$, but not by any Turing machine using an advice of size $f$, when $f \in o(g)$. In this Chapter, we give a detailed proof of this fact. Examination of this proof allows us to improve the result when we bound the size of the advice alphabets. In this case, there are languages that can be decided by a Turing machine with an advice of size $g$, but not by any Turing machine using an advice of size $f$ if $f(n) < g(n)$ for infinitely many integers $n$. Thus, we obtain a finer separation of
the non-uniform complexity classes. This settles a question that was left open by Karp and Lipton[23].

The structure of the Chapter is as follows. First, the concept of advice functions is introduced. Then, we give an example using characteristic strings. Advice functions may be encoded using advice alphabets of different sizes. We give an efficient method to convert advice functions using an advice alphabet to another advice functions using another advice alphabet, such that the advice functions still contain the same information. Next, we introduce non-uniform complexity classes and prove several new hierarchy results.

3.1 Advice Functions

Many sequences are of such a nature that there is no uniform way to describe them. This is because there are uncountably many sequences and only countable many uniform descriptions. Thus, we may have to describe sequences by listing each item in the sequence separately. It is convenient to make a distinction between a common uniform part, and the part which makes it impossible to give a short description: the non-uniform part.

Karp and Lipton[22] introduced the concept of advice functions to formalize this distinction. An advice function is a function from the positive integers to the set of strings. Given a sequence, the advice function maps each integer \( n \) to the description of the \( n \)-th element in a sequence (which is a string). To obtain the sequence, we also need a program to turn the description into the element. Thus, we have split the sequence into a uniform part (the program) and a non-uniform part (the advice function). Just as the original sequence, there may be no way to describe the advice function besides listing each of its values.

By moving the uniform part into the program, one can try to make the non-uniform part as small as possible. The size of the advice function can be seen as a measure of the amount of non-uniformity of a sequence. The size functions induce a partial order on the class of sequences. We say that one sequence is more non-uniform than another if the size function of the first sequence is larger than the size function of the second, a concept that is related to the amount of randomness of a string, see Li and Vitányi[31] for details.

3.1.1 Non-uniformly Defined Languages

In the context of the publication by Karp and Lipton[22], a language corresponds to a sequence: the \( n \)-th item in the sequence consists of the strings of length \( n \) belonging to the language. A program uses its advice function to decide for strings of length \( n \) whether to accept or reject them. The following notation is introduced by Karp and Lipton[22].

**Definition 3.1.** Let \( L \subseteq \Sigma^* \) be a language and \( \alpha : \mathbb{N} \to \Omega^* \) an advice function. Define the set

\[
L: \alpha = \{ x \in \Sigma^* \mid \langle x, \alpha(|x|) \rangle \in L \} .
\]  

(3.1)

Let \( C \) be a class of languages and \( F \) a class of integer-valued functions. Define the non-uniform class

\[
\text{non-uniform class}
\]
3.1 Advice Functions

\[ \mathcal{C}/\mathcal{F} = \{ L: \alpha \mid L \in \mathcal{C}, |\alpha| \in \mathcal{F} \} . \]  \hspace{1cm} (3.2)

We write \( \mathcal{C}/g \) for the class \( \mathcal{C}/\{ h \mid h \text{ is bounded by } g \} \).

If \( L \) is decided by a machine \( M \), then we say that \( M \) decides \( L: \alpha \) with advice function \( \alpha \). If \( \alpha \) is bounded by an integer-valued function \( g \), then we say that \( M \) uses an advice of size \( g \). We call \( \Sigma \) the input alphabet for \( M \) and \( \Omega \) the advice alphabet. We assume that \( M \) uses the tape alphabet \( \Sigma \). Thus, \( \Omega \) is usually a subset of \( \Sigma \). The idea is that \( M \) has a list of advices, one for each length. If \( M \) gets an input \( x \), then \( M \) determines the length of \( x \). Now \( M \) can use the extra information encoded into the advice string for this length to compute the correct output.

**Remark 3.2.** To decide if a string \( x \) belongs to the language \( L: \alpha \), a Turing machine that decides \( L \) takes the tuple \( \langle x, \alpha(|x|) \rangle \) on its input tape. It is clear that the input for the Turing machine consists of both \( x \) and \( \alpha(|x|) \). As a consequence, the time and space measures of a machine with an advice mechanism, which are both defined as a function of the input length, are both dependent on the length of the advice.

### 3.1.2 Characteristic Strings

Next, we use characteristic strings as an advice function. Using such an advice function, any language can be decided with the correct advice. Characteristic strings also proves useful to establish a hierarchy of non-uniform complexity classes.

**Definition 3.3.** Let \( \Sigma \) be an alphabet of size \( c \) and \( n \) an integer. Let \( x_1, \ldots, x_{cn} \) be an enumeration of all strings in \( \Sigma^n \). Let \( L \) be a subset of \( \Sigma^n \). A characteristic string for \( L \) is a binary string \( w \) such that \( w_i = 1 \) iff \( x_i \in L \).

**Remark 3.4.** Note that the definition says nothing about the length of characteristic strings. In some literature, the characteristic string of \( L \) is a characteristic string of length \( cn \). We denote it as the standard characteristic string.

Let \( l \) be the largest integer such that \( x_l \in L \). Then the length of a characteristic string for \( L \) has length at least \( l \). Furthermore, \( w_i = 0 \) for all \( i > l \). The shortest characteristic string for \( L \) has length \( l \).

**Definition 3.5.** We define the Characteristic String (CS) problem for tuples. Consider a tuple \( \langle x, w \rangle \), with \( x \) a string in \( \Sigma^n \) and \( w \) a binary string. Let \( X \) be the subset of \( \Sigma^n \) for which \( w \) is a characteristic string. Then \( \langle x, w \rangle \in CS_{\Sigma} \) iff \( x \in X \).

Let \( L \) be a language over an alphabet \( \Sigma \). Define the advice function \( \alpha \) by letting \( \alpha(n) \) be a characteristic string of \( L \cap \Sigma^n \). It follows that

\[ L = CS_{\Sigma}:\alpha . \]  \hspace{1cm} (3.3)

**Proposition 3.6.** Any language can be decided with an advice of exponential size.
Proof. Let $L$ be a language over an alphabet of size $c$. Definition 3.5 and the remarks following it show that $L$ can be decided with an advice function containing characteristic strings for the sets $L \cap \Sigma^n$, for all $n$. The length of the characteristic strings is at most $c^n$. \hfill \Box

Remark 3.7. The size of the advice function depends on the distribution of the strings in the language: the $n$-th advice value need only be as large as the shortest characteristic string of $L \cap \Sigma^n$. This is an optimization from using the standard characteristic string. However, if $L$ contains the last element of the enumeration, this optimization still yields exponential advice length.

It turns out that for every optimization scheme there are languages that cannot be decided with advice of less than exponential size. This follows directly from Theorem 3.16.

We use the set $CS_\Sigma$ to establish some complexity results. These results can be applied to all decidable classes that contain $CS_\Sigma$. Thus, it is useful to know which classes contain $CS_\Sigma$. The following Proposition shows us which complexity classes contain the set.

Proposition 3.8. The language $CS_\Sigma$ can be recognized in linear time.

Proof. We will construct a Turing machine $M$ that recognizes $CS_\Sigma$. Let $\langle x, w \rangle$ be an input to $M$. Let $|x| = n$ and $|w| = m$. The machine uses two work tapes to store $x$ and $w$ plus an extra work tape. If $w$ contains a symbol other than 0 or 1, the input is rejected. Similarly, $x$ must be in $\Sigma^n$. After separating the input tuple, we ignore the actual input tape and call the tape containing $x$ the “input tape” and the tape containing $w$ the “advice tape”.

Now, $M$ will try to find the position of $x$ on the advice tape to determine the outcome. If $w$ is not long enough, then $M$ will reject the tuple.

Let $c$ be the size of $\Sigma$. For $0 \leq i \leq n$, let $k_i = 1 + \sum_{j=1}^{i} x_j c^{j-1}$. We need to determine the value of $w_{k_n}$ to decide whether $\langle x, w \rangle \in CS_\Sigma$. The machine works in stages. At the beginning of stage $i \geq 1$, the following invariants hold:

- The head of the input tape is on cell $i$;
- the head of the advice tape is on cell $k_{i-1}$;
- the work tape is of length $i - 1$.

In stage $i$, the machine reads the symbol $x_i$ and moves the head of the advice tape $x_i \cdot c^{i-1}$ steps to the right, using the work tape of length $i - 1$. This can be done in $O(c^i)$ steps by generating all strings of length $i - 1$ on the work tape, and moving the advice head $x_i$ steps for every generated string. If the advice head tries to move past the end of $w$, the input is rejected. After this, the advice head will be at position $k_i$. Then, $M$ moves the input head one step to the right and writes a symbol to the work tape to increase the tape length by one. It follows that the invariants hold at the beginning of each stage.

When the input head reaches the blank after $x$, then the advice head is at position $k_n$, which is the position of $x$ on $w$. If the symbol under the advice head is 1, then $M$ accepts, otherwise $M$ rejects the input.
We need $O(n + m)$ steps to extract $x$ and $w$ from the input. The number of steps per stage is about the number of moves to the right on the advice tape (but at least one). This means that the total number of moves to the right is not more than $m$. Hence the total time needed is $O(n + m)$, so $M$ works in linear time.

3.1.3 Conversion between Different Alphabets

When machines are allowed to use advice over alphabets of different sizes, the advice functions can be converted from one alphabet to the other. A string over an alphabet of size $b$ can be considered as a number in base $b$. Converting this number to a different base $d$ will result in a string over an alphabet of size $d$. The original string can be recovered by converting the number in base $d$ back to the number in base $b$. This can be done in quadratic time, as Proposition 3.9 shows.

**Proposition 3.9.** Let $\Omega$ be an alphabet of size $d > 1$ and $\Theta$ an alphabet of size $b > 1$. Let $w$ be an $\Omega$-string. Then $w$ can be turned into a $\Theta$-string $v$ such that $\sum_{i=1}^{\lfloor |w|/d \rfloor} w_i d^{i-1} = \sum_{j=1}^{\lfloor |v|/\Theta \rfloor} v_j \Theta^{j-1}$, in quadratic time, using linear space. The length of $v$ is at most $\lceil (\log d/ \log b) |w| \rceil$.

**Proof.** First, we copy the input $w$ to a work tape. We give the algorithm in pseudo code, and leave the construction of a Turing machine to the reader. See Algorithm 3.1. Let $n$ be the size of $w$, let $s_0 = \sum_{i=1}^{n} w_i d^{i-1}$. In the $j$-th execution of the while loop, the algorithm calculates the unique $s_j$ and $v_j$ such that $s_{j-1} = b \cdot s_j + v_j$ and $0 \leq v_j < b$, and stores $s_j$ in base $d$ in the array that previously held $w$. The algorithm continues until $s_j$ becomes 0, which happens after $m$ steps. Since $v_j b^{j-1}$ equals $s_{j-1} b^{j-1} - s_j b^j$ and $s_m = 0$, it follows that

$$\sum_{j=1}^{m} v_j b^{j-1} = s_0 b^0 - s_m b^m = s_0.$$  \hspace{1cm} (3.4)

The left-hand side of 3.4 can become as large as $b^m - 1$, while $\sum_{i=1}^{n} w_i d^{i-1} \leq d^n - 1$. The smallest value of $m$ for which $b^m \geq d^n$ is $(\log d/ \log b)n$. Thus $m \leq \lceil (\log d/ \log b)n \rceil$.

Each execution of the while loop costs $O(n)$ time, since the tests and the arithmetical operations can all be implemented in constant time, using the transition function of the resulting Turing machine. Since $m$ is in $O(n)$, the conversion can be done in quadratic time. The only space that is needed is the tape to store the values of $s_j$, thus linear space is sufficient.

Suppose $d = a^k$ and $b = a^l$ for an integer $a$. Then a Turing machine can read $l$ digits of $w$ and convert them into $k$ digits of $v$. In this way, the machine can convert any $d$-ary number to base $b$ in linear time.

We define a generalized version of $CS$.

**Definition 3.10.** Consider a tuple $\langle x, w \rangle$, with $x \in \Sigma^n$ and $w$ a string in $\Omega$. Let $v$ be the binary string such that $\sum_{i=1}^{\lfloor |w|/2 \rfloor} v_i 2^{i-1} = \sum_{j=1}^{\lfloor |w|/\Omega \rfloor} w_j \Omega^{j-1}$. Then $\langle x, w \rangle \in CS_{\Sigma, \Omega}$ iff $\langle x, v \rangle \in CS_{\Sigma}$. 
Algorithm 3.1. An algorithm to convert $d$-ary numbers to base $b$.

Note that $CS_{\Sigma} = CS_{\Sigma,(0,1)}$. Observe that $CS_{\Sigma,\Omega}$ can be decided in quadratic time. If the size of $\Omega$ is a power of two, then $CS_{\Sigma,\Omega}$ can be decided in linear time.

Consider the following Property of classes of languages.

Property 3.11. Let $C$ be a class of languages and $L$ a language. Suppose $L = L':\alpha$ for a language $L' \in C$ and an advice function $\alpha$ over an alphabet of size $d$. Let $\beta$ be an advice function over an alphabet of size $b$, such that for every $n$ the string $\beta(n)$ is obtained by converting $\alpha(n)$ to base $b$. Then, there is a language $L'' \in C$ such that $L = L'':\beta$.

For example, the class $P$ satisfies this Property.

3.2 Non-uniform Complexity Classes

It is intuitive to assume that as machines have access to larger advice functions, they are able to decide languages that were previously undecidable. In accordance with this intuition, Karp and Lipton[22] claimed that $P/f \subset P/g$ when $f(n) < g(n)$ holds infinitely often. In a later publication[23], the statement was weakened
to $P/f \subseteq P/g$ (which is true by definition). Hermo and Mayordomo[18] proved that the inclusion is proper when $f$ in $o(g)$. They used notions of Kolmogorov Complexity to arrive at this conclusion. We will give a different proof, which will give us more insight into the sizes of the involved classes. This allows us to improve the result when we restrict the size of the advice alphabet.

### 3.2.1 A Technical Result

In this and the following subsections, we will give some results which share the same idea. To minimize the number of repetitions, we will state the idea in a separate, technical Lemma.

Let $\Sigma$ be an input alphabet and $\Omega$ an advice alphabet. Let $g$ be an integer-valued function. Our goal is a statement of the form: For suitable functions $f$ and integers $n$, a machine using advice of size $g$ and an advice alphabet $\Omega$ can decide more different subsets of $\Sigma^n$ than any machine using advice of size $f$. When this is true, we can use a diagonalizing argument to construct a language that cannot be decided with advice $f$ (see Lemma 3.14).

**Proposition 3.12.** Let $\Sigma$ be an alphabet of size $c$ and $\Omega$ an alphabet of size $d \geq 2$. Let $g$ be an integer-valued function such that $g(n) \leq c^n / \log d$ for all $n$. Then, a Turing machine for $CS_{\Sigma,\Omega}$ can decide $d^{g(n)}$ different subsets of $\Sigma^n$ with advices of size $g$.

**Proof.** Since there are no more than $2^{c^n}$ subsets of $\Sigma^n$, a characteristic string corresponds to a unique subset iff it has length $c^n$ or less.

Any string $w$ of length $g(n)$ over the alphabet $\Omega$ can be converted into a binary string $v$ of length $\lceil (\log d)g(n) \rceil$ (see Proposition 3.9). Thus, $v$ is a characteristic string of length at most $c^n$. It follows that $v$ corresponds to a unique subset of $\Sigma^n$. The string $w$ corresponds to the same subset. $\square$

**Proposition 3.13.** Let $\Sigma$ be an alphabet of size $c$ and $\Theta$ an alphabet of size $b$. Let $h$ be an integer-valued function. Then, an arbitrary Turing machine can decide at most $\sum_{i=0}^{h(n)} b^i$ different subsets of $\Sigma^n$ with advices of size at most $h$ over the alphabet $\Theta$.

**Proof.** Since there are only $b^i$ different advice values of length $i$ over the alphabet $\Theta$ for every $0 \leq i \leq h(n)$, the result follows immediately. $\square$

Let $M$ be an arbitrary Turing machine using an advice alphabet of size $b$. Consider the following inequality.

$$\sum_{i=0}^{h(n)} b^i < d^{g(n)} . \quad (3.5)$$

When it holds, there is a subset of $\Sigma^n$ that can be decided by a Turing machine for $CS_{\Sigma,\Omega}$ with an advice of size $g$ over an alphabet $\Omega$ of size $d$, but not by $M$ with an
advice of size at most $h$. Inequality (3.5) is the basis for the next results. Basically, for every combination of Turing machine and advice size, a suitable integer $n$ for which (3.5) holds has to be found.

With these facts in place, we can give the technical Lemma. It can be viewed as a recipe for the actual Theorems which are given later.

**Lemma 3.14.** Suppose the following conditions all hold.

- Let $\Sigma$ be an input alphabet of size $c$.
- Let $\Omega$ be an advice alphabet of size $d \geq 2$.
- Let $\mathcal{B}$ be a class of allowed advice alphabet sizes.
- Let $g$ be an integer-valued function.
- Let $\mathcal{F}$ be a class of integer-valued functions.
- Let $\mathcal{H}$ be a countable class of integer-valued functions.
- Let $N$ be an integer.
- Finally, let $L$ be a language that can be decided by a Turing machine for $CS_{\Sigma, \Omega}$ with an advice of size $g$.

Suppose that $g(n) \leq c^n/\log d$ for every $n$. Suppose that for every $f \in \mathcal{F}$ and every $b \in \mathcal{B}$ there is a function $h \in \mathcal{H}$ such that $f(n) \leq h(n)$ for all but finitely many integers $n$ and (3.5) holds for $h$ and $b$, for infinitely many integers $n$. Then, there is a language $L'$ that can be decided by a Turing machine for $CS_{\Sigma, \Omega}$ with an advice of size $g$ over the advice alphabet $\Omega$, but not by any Turing machine with an advice of size bounded by a function $f \in \mathcal{F}$ over an alphabet in $\mathcal{B}$. Furthermore, $L' \cap \Sigma^n = L \cap \Sigma^n$ for all $n \leq N$.

**Proof.** Let $\alpha$ be the $g$-bounded advice used to decide $L$. Consider the class of all tuples of the form $(h, b, m, M)$ for functions $h \in \mathcal{H}$, integers $b$ and $m$ and Turing machines $M$, such that (3.5) holds for $h$ and $b$, for infinitely many integers $n$. Observe that this class is countable, so there is an enumeration of all its tuples. To each tuple $(h, b, m, M)$, an integer $n > m$ is assigned such that (3.5) holds for $h$ and $b$ and this $n$. The integer $n$ is chosen such that $n$ is larger than $N$ and larger than the integers that were assigned to previous tuples in the enumeration. This is possible since there are infinitely many of such integers $n$ for every $h$ and $b$.

By combining (3.5) and Propositions 3.12 and 3.13, it follows that there is an advice value $\alpha'(n)$ of size $g(n)$ that helps a Turing machine for $CS_{\Sigma, \Omega}$ to decide a subset $L_n$ of $\Sigma^n$ that cannot be decided by $M$ with any advice of size bounded by $h$ over an alphabet of size $b$.

The advice function $\alpha'$ is constructed in this way, with $\alpha'(n) = \alpha(n)$ if $n$ is not assigned to a tuple. It follows that $\alpha'$ is of size $g$. Let $L'$ be the language that a Turing machine for $CS_{\Sigma, \Omega}$ decides with advice $\alpha'$. Note that $L' \cap \Sigma^n = L_n$ if $n$ is assigned to a tuple and $L' \cap \Sigma^n = L \cap \Sigma^n$ otherwise. In particular, $L' \cap \Sigma^n = L \cap \Sigma^n$ for all $n \leq N$.

Suppose that $L'$ can be decided by a Turing machine $M$ with an advice $\beta$ of size bounded by a function $f \in \mathcal{F}$ over an alphabet of size $b \in \mathcal{B}$. Let $h \in \mathcal{H}$ be a function and $m$ an integer such that $f(n) \leq h(n)$ for all $n > m$ and (3.5) holds for $h$ and $b$, for infinitely many integers. Let $n$ be the integer assigned to the tuple $(h, b, m, M)$. The Turing machine $M$ uses the advice string $\beta(n)$ to recognize a subset $S = L' \cap \Sigma^n$. Since $|\beta(n)| \leq f(n)$ and $n > m$, it follows that $|\beta(n)| \leq h(n)$.
Since (3.5) holds for $h$ and $b$ and this $n$, the set $L_n$ is different from $S$. But by construction, $L' \cap \Sigma^n$ equals $L_n$. This yields a contradiction. So $L'$ cannot be decided by any Turing machine using an advice of size bounded by a function $f \in \mathcal{F}$ over an alphabet of size $b \in \mathcal{B}$.

The existence of a language $L'$ in Lemma 3.14 was proved by constructing a single language. Here, we will show that there are in fact many of such languages. Recall the distance function on the class of languages from Chapter 2. A subset of languages is dense in this class if for every language $L$ and every $\epsilon > 0$, we can find a language in the subset with distance less than $\epsilon$ to $L$.

**Proposition 3.15.** Suppose the following conditions all hold.

- Let $\Sigma$ be an input alphabet of size $c$.
- Let $\Omega$ be an advice alphabet of size $d \geq 2$.
- Let $\mathcal{B}$ be a class of allowed advice alphabet sizes.
- Let $g$ be an integer-valued function.
- Let $\mathcal{F}$ be a class of integer-valued functions.
- Let $\mathcal{H}$ be a countable class of integer-valued functions.

Suppose that $g(n) \leq c^n / \log d$ for every $n$. Suppose that for every $f \in \mathcal{F}$ and every $b \in \mathcal{B}$ there is a function $h \in \mathcal{H}$ such that $f(n) \leq h(n)$ for all but finitely many integers $n$ and (3.5) holds for $h$ and $b$, for infinitely many integers $n$. Let $\mathcal{G}$ be the class of integer-valued functions such that $g' \in \mathcal{G}$ iff $g'(n) \leq g(n)$ holds for all but finitely many $n$. Let $\mathcal{L}$ be the class of languages such that $L' \in \mathcal{L}$ iff $L'$ can be decided by a Turing machine for $CS_{\Sigma,\Omega}$ with an advice of size $g' \in \mathcal{G}$ over the advice alphabet $\Omega$, but not by any Turing machine with an advice of size bounded by a function $f \in \mathcal{F}$ over an alphabet in $\mathcal{B}$. Then $\mathcal{L}$ is dense in the class of languages over the alphabet $\Sigma$.

**Proof.** Let $S$ be an arbitrary language over the alphabet $\Sigma$ and let $\epsilon > 0$. Define $m = \lceil 1/\epsilon \rceil$. We let $L$ be the finite set containing the strings in $X$ of length $m$ or less. Observe that $L$ can be decided by a Turing machine for $CS_{\Sigma,\Omega}$ with an advice function that is bounded by a function in $\mathcal{G}$. Apply Lemma 3.14 with $N \geq m$. Then, we obtain a language $L' \in \mathcal{L}$. By construction of $L$, it follows that $L' \cap \Sigma^n = L \cap \Sigma^n = X \cap \Sigma^n$ for all $n \leq m$. Thus, $L'$ is a language in $\mathcal{L}$ with a distance of less than $\epsilon$ to $S$.

It follows that $\mathcal{L}$ is dense in any subclass of subsets of $\Sigma^*$. As a result, the non-uniform complexity classes are evenly distributed in the class of languages, i.e., it is not possible to find a small open set of languages that contains no languages of a given non-uniform complexity. Another conclusion is that there are infinitely many languages to prove the results of the next sections.

### 3.2.2 Advice Alphabets of Unbounded Size

Turing machines may have arbitrarily large tape alphabets. Thus, the sizes of advice alphabets that machines can use can grow equally large. This implies that...
arbitrary amounts of information can be encoded into advice functions by using large enough advice alphabet sizes. Re-examining (3.5), it becomes clear that the inequality holds when \( \frac{1}{b} h(n) + 1 - 1 < d(n) \). Therefore, \( h(n) \) should be at most \( (\log d/ \log b) g(n) - 1 \). Since the advice alphabet size \( b \) depends on the machine in the \( n \)-th tuple of the enumeration, it follows that \( b \) is a function of \( n \). Thus, it becomes clear that \( f \) must be in \( o(g) \). Theorem 3.16, originally observed by Hermo and Mayordomo[18] with an argument from Kolmogorov complexity theory, follows from this result.

**Theorem 3.16 (Hermo–Mayordomo).** Let \( D \) be any recursive class containing \( \text{DTIME}(n) \) and let \( C \) be the class of all recursive languages. Let \( F \) and \( G \) be classes of integer-valued functions such that there is a function \( g \in G \) with \( g \in o(2^n) \) and \( f \in o(g) \) for every \( f \in F \). Then \( D/G \) is not included in \( C/F \).

**Proof.** Let \( \Sigma \) and \( \Omega \) be binary alphabets. Let \( B \) be the set of positive integers. Consider the countable class of integer-valued functions

\[
\mathcal{H} = \left\{ \left(\frac{b}{\log b}\right)^{-1} \cdot g(n) - 1 \mid b \geq 2 \right\}.
\]  

(3.6)

Let \( N = 0 \) and let \( L = \emptyset \). Observe that \( L \) can be decided with an advice of size 0. Note also that \( g(n) \leq 2^n / \log 2 \) for every \( n \).

Let \( f \) be a function in \( F \) and \( b \) an integer in \( B \). Let \( h \in \mathcal{H} \) be the function defined by \( h(n) = \left(\frac{b}{\log b}\right)^{-1} \cdot g(n) - 1 \) (or \( h(n) = g(n) - 1 \) if \( b = 1 \)). Observe that (3.5) holds for \( h \) and \( b \), for all integers \( n \). Since \( f \in o(g) \), it follows that \( f(n) \leq h(n) \) for all but finitely many integers \( n \).

Thus, we may apply Lemma 3.14 to obtain a language \( L' \) that is decided by \( \mathcal{C}_\Sigma \) with a binary advice of size \( g \), but not by any Turing machine with an advice of size bounded by a function \( f \in F \) over an alphabet of any size. Since \( \mathcal{C}_\Sigma \in \text{DTIME}(n) \), it follows that \( L' \) is in \( D/G \), but not in \( C/F \).

\( \square \)

**Theorem 3.17.** Let \( \Sigma \) be an input alphabet of size \( c \). Let \( g \) be a function such that \( g(n) \leq c^n \). Let \( C \) be a class of decidable languages. Let \( F \) be a countable class of integer-valued functions such that \( f \notin \Omega(g) \) for every function \( f \in F \). Then \( \{\mathcal{C}_\Sigma\}/g - C/F \neq \emptyset \).

**Proof.** Let \( B \) be the set of positive integers. Let \( \mathcal{H} \) be the class \( F \). Let \( N = 0 \) and \( L = \emptyset \). Observe that \( g(n) \leq c^n / \log 2 \).

Let \( f \) be a function in \( F \) and \( b \) an integer in \( B \). Since \( f \notin \Omega(g) \), there are infinitely many integers \( n \) such that \( f(n) \leq \left(\frac{b}{\log b}\right)^{-1} g(n) - 1 \). It follows that (3.5) holds for \( f \) and \( b \), for infinitely many integers \( n \). Since \( \mathcal{H} = F \), there is a function \( h \in \mathcal{H} \), i.e., the function \( h = f \), such that \( f(n) \leq h(n) \) holds for all but finitely many integers \( n \).

Thus, we can apply Lemma 3.14. It follows that there is a language \( L' \) that is in \( \{\mathcal{C}_\Sigma\}/g \), but not in \( C/F \).

\( \square \)

**Corollary 3.18.** Let \( f \) and \( g \) be integer-valued functions such that \( f \notin \Omega(g) \). Then \( P/f \) is a proper subset of \( P/g \).
Proof. Follows from Theorem 3.17.

\[\square\]

**Corollary 3.19.** Let \( f \) and \( g \) be integer-valued functions such that \( f \notin \Omega(g) \). Then \( P/g - P/f \) is dense in \( P/g \).

Proof. Let \( \mathcal{G} \) be the class of integer-valued functions such that \( g'(n) \leq g(n) \) for all but finitely many integers \( n \). It follows from Proposition 3.15 that \( P/\mathcal{G} - P/f \) is dense in \( P/g \). Since an advice function bounded by a function \( g' \) in \( \mathcal{G} \) can encode only a finite amount of extra information compared to an advice function of size \( g \), this extra information can be stored in the finite control of a Turing machine instead. Thus, \( P/\mathcal{G} \) is a subclass of \( P/g \). Hence, \( P/g - P/f \) is dense in \( P/g \).

\[\square\]

Karp and Lipton[22] originally claimed that \( P/f \) is a proper subset of \( P/g \) if \( f(n) < g(n) \) holds for infinitely many integers \( n \). Corollary 3.18 is somewhat weaker than this claim. However, for the class \( P \), the separation cannot be improved, as the following Proposition shows.

**Proposition 3.20.** Let \( \mathcal{C} \) be a class of languages satisfying Property 3.11. Let \( g \) be an integer-valued function and \( \mathcal{F} \) be a class of integer-valued functions with a function \( f \in \mathcal{F} \) such that \( f \in \Omega(g) \). Then \( \mathcal{C}/g \subseteq \mathcal{C}/\mathcal{F} \).

Proof. Let \( f \in \mathcal{F} \) be a function such that \( f \in \Omega(g) \). Let \( N \) and \( m \) be integers such that \( f(n) \geq m^{-1} \cdot g(n) \) for all \( n \geq N \). Suppose \( L \) is a language in \( \mathcal{C}/g \). Let \( M \) be a Turing machine that decides \( L \) with an advice \( \alpha \) of size bounded by \( g \). Let \( b \) be the size of the advice alphabet that \( M \) uses. If \( b > 1 \), then we can encode the advice \( \alpha \) using an alphabet of size \( b^m \). This way, we can decrease the length of the advice by a factor of \( m \) (see Proposition 3.9). If \( b = 1 \), then we can encode the advice with an alphabet of size \( 2m \), which decreases the length of the advice logarithmically. In both cases, the encoded advice is bounded by \( f \), which means that \( L \) can be decided by a Turing machine using an advice bounded by \( f \) that first decodes the advice and then simulates \( M \) on the tuple of input and advice value.

\[\square\]

Observe that the decoding used in the proof can be done in linear time. This Proposition shows that Theorem 3.17 cannot be improved for countable classes of functions without further restrictions on the allowed advice alphabet sizes.

**Corollary 3.21.** Let \( f \) and \( g \) be integer-valued functions such that \( f \in \Omega(g) \). Then \( P/g \) is a subset of \( P/f \).

Proof. Follows from Proposition 3.20 and the fact that \( P \) satisfies Property 3.11.

\[\square\]
3.2.3 Advice Alphabets of Bounded Size

In the statement of Theorem 3.16, we made no restriction on the allowed advice alphabet size, which implied that there were no bounds on the sizes. If we restrict the possible sizes, then we can improve the result of the Theorem. Since hardware implementations impose practical bounds on the sizes of the alphabets used by machines, this assumption is not unreasonable.

**Theorem 3.22.** Let $\Sigma$ be an input alphabet of size $c$ and $\Omega$ an advice alphabet of size $d \geq 2$. Let $g$ be an integer-valued function such that $g(n) \leq c^n/\log d$. Let $\mathcal{F}$ be a countable class of integer-valued functions such that for every function $f \in \mathcal{F}$, the inequality $f(n) \leq g(n) - 1$ holds infinitely often. Then, there is a language that can be decided by a Turing machine with an advice of size $g$ over the alphabet $\Omega$, but not by any Turing machine using an advice bounded by $f$ over an advice alphabet of size $d$ or less.

**Proof.** Let $\mathcal{B} = \{1, \ldots, d\}$. Let $\mathcal{H}$ be the class $\mathcal{F}$. Let $N = 0$ and $L = \emptyset$.

Let $f$ be a function in $\mathcal{F}$ and $b$ an integer in $\mathcal{B}$. Since $f(n) + 1 \leq g(n)$ holds infinitely often and $b \leq d$, it follows that (3.5) holds for $f$ and $b$, for infinitely many integers $n$. Note that $f \in \mathcal{H}$.

Thus, we can apply Lemma 3.14, to obtain a language $L'$ that can be decided by a Turing machine for $CS_{\Sigma,\Omega}$ with an advice of size $g$ over the alphabet $\Omega$, but not by any Turing machine using an advice bounded by a function $f$ in $\mathcal{F}$ over an alphabet of size $d$ or less.

Thus, the original statement made by Karp and Lipton[22] holds if the advice alphabets are bounded in size. For instance, when using only binary advices, $P/f$ is a proper subset of $P/g$ whenever $f(n) < g(n)$ holds infinitely often and $g$ is exponentially bounded. The statement remains true if e.g. a 10, 16 or 26 letter alphabet are used.

**Corollary 3.23.** Suppose Turing machines may only use advice functions with advice alphabets of size bounded by an integer $d \geq 2$. Let $c$ be an integer and let $f$ and $g$ be integer-valued functions such that $f(n) < g(n)$ holds infinitely often and $g(n) \leq c^n/\log d$ for all $n$. Then $P/f$ is a proper subset of $P/g$. Furthermore, $P/g - P/f$ is dense in $P/g$.

**Proof.** This follows from Theorem 3.22. The last statement follows from Proposition 3.15 (see also the proof of Corollary 3.19).

This non-uniform separation cannot be improved further, since any finite amount of extra information in the advice function could also be coded into the machine itself.
3.3 Conclusions

Simple counting arguments, the possibility to enumerate all possible interesting combinations of machine and advice and diagonalizing arguments are the ingredients that were essential to the central theorems of the Chapter. These three techniques are very useful and will be used multiple times in this thesis. To enumerate all the interesting cases, the number of interesting cases should be countable. This can in some cases be a restriction, but we can often deploy a countable subset of interesting cases which covers all interesting cases in some way. An example of this is seen in the use of the classes $F$ and $H$ in Theorem 3.16. Here, $F$ was a possibly uncountable class of integer-valued functions used as advice bounds. The class $H$ was constructed such that every interesting combination of advice bound $f \in F$ with a machine was covered by the combinations of all functions from $H$ with the same machine.

Hermo and Mayordomo[18] gave a proof for Theorem 3.16 that is based on Kolmogorov complexity theory. We should note here that the foundations of Kolmogorov complexity theory rely on the same kind of counting arguments. The role of the advice alphabet sizes was obvious in (3.5), while the use of Kolmogorov complexity theory did not make clear that the results could be improved by restricting the allowed sizes of the advice alphabets. This illustrates that while abstract theories can greatly reduce the complexity of proofs, one should not be afraid to get one's hands dirty in order to obtain improvements over the results that an abstract framework so easily provides.

For advice sizes, the picture is now complete. Hermo and Mayordomo[18] proved that $P/F$ is properly contained in $P/g$ if $f \in o(g)$ for every $f \in F$. We showed that $P/g$ is contained in $P/F$ if there is a function $f \in F$ such that $f \in \Omega(g)$. Karp and Lipton[22] originally claimed that $P/f$ was properly contained in $P/g$ if $f(n) < g(n)$ for infinitely many integers $n$. While we showed that this is impossible for unbounded advice alphabet sizes, we also showed that the claim holds if the allowed advice alphabets are bounded by a constant. On the other hand, if $f(n) \geq g(n)$ for all but finitely many integers $n$, then the advices for the finitely many integers $n$ for which $f(n) < g(n)$ can be encoded into the finite control of a Turing machine. Such a machine can then use an advice of length $f$ to decide languages that need an advice of length $g$. So in this case, $P/g$ is contained in $P/f$.

The careful reader may have noticed that the advices of size $g$ were all over an alphabet with more than one letter. For the unbounded case, this is crucial, since a one-letter advice of size $g$ can be encoded into a binary string of length $O(\log g)$. However, if machines may only use advices over one-letter alphabets, then Theorem 3.22 remains valid. In this case, the counting argument can be simplified by observing that there are $f(n)$ different advices bounded in length by $f$ and $g(n)$ different advices bounded in length by $g$.

The class $F$ in Theorem 3.22 is a countable class. This is necessary to enumerate all the interesting cases. While the result can be changed to include classes of functions that are bounded by a countable class of functions, it remains an open question if the result holds for all classes of functions $F$ such that $f(n) < g(n)$ for infinitely many integers $n$ for every $f \in F$. 