A natural way to model evolving systems is to use sequences. Sequences play an important role in this thesis. In this Chapter, we will take a closer look at sequences of machines and their relation to advice functions. The machines that we consider can be of any fixed type, ranging from finite automata to Turing machines or other models of computation.

Let $\mathcal{M}$ be a class of machines with an input alphabet $\Sigma$. Let $M_1, M_2, \ldots$ be a sequence of machines in $\mathcal{M}$. We view such a sequence of machines as one single computing entity which has the ability to decide subsets of $\Sigma^*$ by utilizing the components of the sequence. A set $L$ is said to be decided by the sequence iff machine $M_n$ decides $L \cap \Sigma^n$ for every $n$.

We classify sequences of machines based on the description sizes of the machines that make up the sequence. Using this complexity measure, we define complexity classes for sequences of machines. Our first goal is to state a theorem which relates the complexity of a sequence of machines to the complexity of a single machine with an advice mechanism of a suitable complexity. An example of such a theorem was first given by Karp and Lipton[22], who showed that the class of languages decided by (sequences of) polynomially sized Boolean circuits equals the class of languages decided by a Turing machine with polynomial advice. We will state a more general version of the theorem, i.e., we don’t fix the machine models to be used in the characterization in advance. The theorem given by Karp and Lipton is then a consequence of this general version. We will give several more instances of the theorem, e.g. for sequences of resource-bounded Turing machines.

In setting up the framework for the result, a conflict of interests arises. On the one hand, we want a theorem that is as general as possible, which means that we can only use aspects of machines which are shared between most (if not all) models of computation. On the other hand, in order to achieve this, we need formal definitions of these aspects in order to actually apply the results. We compromise by defining *meta-properties* of models of machines, which we use as formal definitions.
When we want to apply the equivalence result to actual machine models, we just have to translate these meta-properties into actual properties of the machines, using the definitions of the class of models to which they belong. As an example, we introduce the concept of a machine calling another machine, which is implemented by Turing machines by integrating the transition function of the second into the transition function of the first. These meta-properties build on the concepts that were introduced in Chapter 2.

The Chapter begins with a brief introduction to the fundamental properties of machine models. Using these properties, the general theorem is given. The theorem states the relationship between sequences of machines and machines with an advice mechanism. After this, the theorem is applied to several well-known and often used machine models. Next, we explore the possibilities of using different ways of describing machines as advice functions. By using different descriptions, we can generate more efficient or more powerful advice functions, depending on the results we want to achieve. This idea is applied to show several more equivalences where we specifically aim at sequence-based characterizations of the classes $\text{LOGSPACE/poly}$ and $\text{P/poly}$, within our general framework. Then, one of the fundamental properties is explored in more detail, and a more concrete property is defined which implies this fundamental property. Finally, some conclusions are given for the theory of sequences in the context of modeling evolving systems.

### 4.1 General Aspects of Machines

Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be machine models. Given a sequence of machines of type $\mathcal{M}_1$, we will construct a machine $M$ of type $\mathcal{M}_2$ and an advice function, such that $M$ uses the advice function to decide the same language as the sequence of machines. This implies that $M$ should be able to handle multiple input lengths. We assume that $\mathcal{M}_1$ and $\mathcal{M}_2$ satisfy certain conditions, listed as Properties 4.1 and 4.2.

**Property 4.1.** Given a sequence of machines of type $\mathcal{M}_1$, there is an encoding mechanism $e$ for the machines in the sequence and a machine $M$ of type $\mathcal{M}_2$ that takes as input any string $x$ and as advice any string $w$, as long as $w$ is the description of a machine in the sequence, such that $M$ simulates the operation of the machine encoded by $w$ on the input $x$. Furthermore, there are functions $T, S : \mathbb{N}^5 \rightarrow \mathbb{N}$ such that if the machine encoded by $w$ is of size $m'$ and runs in time $T'$ and space $S'$, then $M$ uses $O(T(|x|, |w|, S', T', m'))$ time and $O(S(|x|, |w|, S', T', m'))$ space.

**Property 4.2.** For every machine $M_2$ of type $\mathcal{M}_2$, every integer $n$ and every string $w$, there is a machine $M_1$ of type $\mathcal{M}_1$, such that for every string $x$ of length $n$ the machine $M_1$ simulates the operation of $M_2$ on the input $x$ using advice $w$, if $M_1$ has the strings $x$ and $w$ stored as data. Furthermore, there are functions $m, T, S : \mathbb{N}^5 \rightarrow \mathbb{N}$, such that if $M_2$ is of size $m'$ and uses time $T'$ and space $S'$, then $M_1$ has a size of $O(m(n, |w|, S', T', m'))$ and runs in time $O(T(n, |w|, S', T', m'))$ and space $O(S(n, |w|, S', T', m'))$.

Other important properties of machine models we need are:

**Property 4.3.** Machines of type $\mathcal{M}$ can handle inputs of all lengths.
Property 4.4. Machines of type $\mathcal{M}$ can store any string of length $n$ as mutable data using at most $O(n)$ space.

Property 4.5. Every string of length $n$ can be stored by a machine of type $\mathcal{M}$ as immutable data using at most $O(n)$ size.

Note that storing information in the immutable data of a machine essentially changes the machine itself. Thus, when a string that is stored in the immutable data is altered, the machine that stores the string is altered as well.

Property 4.6. Any machine $M$ of type $\mathcal{M}$ can be called by any other machine $M'$ of type $\mathcal{M}$. The machine $M$ performs the actions on (part of) the contents of the data of the calling machine $M'$. Suppose $M$ has size $m$ and runs in time $T$ and space $S$. If $M'$ calls $M$, then the size of $M'$ is increased by $O(m)$, and $M'$ uses an additional $O(T)$ time and $O(S)$ space.

Definition 4.7. Let $M$ be a machine that has among its data a description of an object $o$. We say that $M$ has a pointer to $o$ if $M$ is in a configuration which contains $o$.

For example, the machine $M$ may have $o$ on a tape, and a tape head positioned at the beginning of $o$. Or, $M$ may have $o$ stored using states, and be in a state that holds $o$. As machine models get more exotic, more examples could be dreamed up.

4.1.1 Sequences of Machines

An important goal of this Chapter is to state a general version of an equivalence result that links sequences of machines to machines with advice. The terminology for machines with advice mechanisms has already been explored in Chapter 3. Here, we define some terminology for sequences of machines.

Definition 4.8. Let $\Sigma$ be an alphabet. A sequence of machines is said to decide a language $L \subseteq \Sigma^*$ if the $n$-th machine in the sequence decides the subset $L \cap \Sigma^n$ as a subset of $\Sigma^n$.

Definition 4.9. A sequence of machines is said to be of size $m$ for an integer-valued function $m$, if the $n$-th machine is of size $m(n)$.

Definition 4.10. Given an encoding mechanism $e$, a sequence of machines is said to have descriptions of size $d$ for an integer-valued function $d$, if the $n$-th machine can be encoded using $e$ within a string of size $d(n)$.

Definition 4.11. A sequence of machines is said to be of time (or space) complexity $T$ for an integer-valued function $T$, if the $n$-th machine uses at most time (or space) $T(n)$ for its inputs of length $n$.

Alternatively, we say that a sequence of machines has a running time (or a space usage) of $T$, or that a sequence of machines runs in time (or space) $T$. 
4.2 A General Equivalence Result

With the terminology of the previous section in place, we can give the general version of the results that links sequences of machines to machines with advice.

**Theorem 4.12.** Let \( M_1 \) and \( M_2 \) be two machine models. Let \( M_1, M_2, \ldots \) be a sequence of machines of type \( M_1 \) with descriptions of size \( f \), using an encoding mechanism \( e \). Suppose that Property 4.1 holds. Let \( L \) be the language decided by the sequence of machines. Then, \( L \) can be decided by a machine \( M \) of type \( M_2 \) with an advice of size \( f \). Furthermore, if machine \( M_n \) runs in time \( T_n \) and space \( S_n \), then \( M \) runs in time \( O(T(n, f(n), S_n, T_n)) \) and space \( O(S(n, f(n), S_n, T_n)) \).

**Proof.** Let \( w_n \) be the description of \( M_n \). By Property 4.1, there is a machine \( M \) of type \( M_2 \) running in the desired time and space that takes any string \( x \) as input and \( w_{|x|} \) as advice and simulates \( M_{|x|} \) on \( x \). Thus, \( M \) accepts \( x \) with advice \( w_{|x|} \) iff \( M_{|x|} \) accepts \( x \) iff \( x \in L \). Hence, \( L \) is decided by \( M \) using the advice function \( n \mapsto w_n \) of size \( f \).

\( \square \)

**Theorem 4.13.** Let \( M_1 \) and \( M_2 \) be two machine models satisfying Property 4.2. Suppose that \( M_1 \) satisfies Property 4.5 and 4.6. Let \( L \) be a language and \( f \) an integer-valued function. If \( L \) can be decided by a machine \( M \) of type \( M_2 \), of size \( m' \) running in time \( T' \) and space \( S' \), with an advice of size \( f \), then \( L \) can be decided by a sequence of machines of type \( M_1 \) of size \( O(f(n) + m(n, f(n), S', T', m')) \). Furthermore, the sequence of machines runs in time \( O(T(n, f(n), S', T')) \) and space \( O(S(n, f(n), S', T')) \).

**Proof.** Let \( M \) be a machine of type \( M_2 \) that decides \( L \) with an \( f \)-bounded advice function \( \alpha \). Suppose \( M \) is of size \( m' \) and runs in time \( T' \) and space \( S' \). Let \( w \) be a string. By Property 4.2, there is a machine \( M_{n,w} \) of type \( M_1 \) such that for every string \( x \) of length \( n \) the machine \( M_{n,w} \) simulates the operation of \( M \) on input \( x \) using advice \( w \), if \( M_{n,w} \) has the strings \( x \) and \( w \) stored as data.

Construct the machine \( M_n \) of type \( M_1 \) as follows: \( M_n \) has the string \( w = \alpha(n) \) stored in its immutable data (using \( O(f) \) size by Property 4.5). Given an input \( x \) of length \( n \), the machine \( M_n \) calls \( M_{n,w} \). By Property 4.6, this can be done with an additional size of \( O(m(n,f(n),S',T',m')) \), in the desired time and space. Since the call on \( M_{n,w} \) has \( x \) and \( w \) stored as data, \( M_n \) accepts \( x \) iff \( M \) accepts \( x \) using advice \( w \). Summing up, the total size needed for \( M_n \) is \( O(f(n) + m) \). Observing that \( M_n \) decides \( L \cap \Sigma^n \) finishes the proof.

\( \square \)

Due to the abstract nature of the results, they cannot be applied straightaway. One must first express the abstract properties of a machine model using the concrete definitions of the model. Doing this differs from one model to the other.

As an example we show how Theorems 4.12 and 4.13 imply Karp and Lipton’s result for sequences of Boolean circuits[22]. The result follows by taking the Boolean circuit model for \( M_1 \) and the Turing machine model for \( M_2 \) and determining the correct functions used in Properties 4.1 and 4.2.
Corollary 4.14 (Karp and Lipton). A language $L$ can be decided by a sequence of polynomially sized Boolean circuits iff $L$ is in $P/poly$.

Proof. We use the standard encoding for Boolean circuits. Then, a circuit of size $m$ can be described using a string of length $O(m \log m)$. The Circuit Value Problem ($CVP$, as defined in Balcázar et al.[1]) takes a binary string $x$ and a description of a Boolean circuit $w$ and determines if $x$ is accepted by the circuit described by $w$. Thus, a Turing machine that solves the $CVP$ satisfies Property 4.1. The $CVP$ can be solved by a Turing machine in quadratic time using linear space (see Balcázar et al.[1]). It follows from Theorem 4.12 that a sequence of polynomially sized circuits can be simulated by a Turing machine in polynomial time using advice of polynomial size.

Conversely, observe that Boolean circuits have no implementation of mutable data, so the space complexity of the sequence can be ignored. Note that Boolean circuits satisfy Property 4.5 and 4.6. It is well-known that a Turing machine running in time $T'$ can be simulated by a Boolean circuit of size $m \in O((T')^2)$, with a depth $T \in O(T')$ (see Balcázar et al.[1]). Thus Property 4.2 is satisfied and Theorem 4.13 can be applied.

However, a Turing machine incorporates the advice in its input, whereas the Boolean circuit has to embed the advice using constant gates. Therefore, a Turing machine running in time $T'$ with an advice of size $f$ can be simulated by a sequence of Boolean circuits of size $O(f(n) + (T'(n+f(n)))^2)$. Since $T'$ and $f$ are polynomial by the assumptions of the Corollary, the sequence is polynomially sized.

4.2.1 Sequences of Resource-Bounded Turing Machines

An interesting example of the equivalence result is the case when $M_1$ and $M_2$ are both taken to be the Turing machine models with suitable complexity measures. First, sequences of uniformly time-bounded Turing machines are considered.

Proposition 4.15. Let $f$ be a integer-valued function. Let $e$ be an encoding mechanism. If a language $L$ is decided by a sequence of Turing machines running in polynomial time (or logarithmic space), with descriptions of size $f$ using the encoding $e$, then $L$ can be decided by a Turing machine running in polynomial time (or logarithmic space), with an advice of size $f$.

Proof. We can use a universal Turing machine $M$ that takes descriptions of Turing machines using the encoding $e$ to satisfy Property 4.1. The machine $M$ uses the descriptions of the machines in the sequence as an advice function, so $M$ uses an advice of size $f$. Suppose the $n$-th machine uses time $T_n$ and space $S_n$. Then $M$ can simulate the $n$-th machine of the sequence in time $T \in O(T_n^2 f(n))$ and space $S \in O(S_n)$.

Observe that $M$ has both the input and the advice on its input tape. Let $n' = n + f(n)$. Then $T$ is polynomially bounded as a function of $n'$ if $T_n$ is polynomial in $n$. Similarly, $S$ is logarithmically bounded in $n'$ if $S_n$ is logarithmic in $n$. Thus, by Theorem 4.12, a sequence of Turing machines running in polynomial time (or logarithmic space), with descriptions of size $f$, can be simulated by a
Turing machine running in polynomial time (or logarithmic space), with an advice of size $f$. 

\[\text{Corollary 4.16.} \text{ Let } m \text{ be an integer-valued function and } k \text{ an integer. If a language } L \text{ is decided by a sequence of } k\text{-tape Turing machines of size } f \text{ running in polynomial time (or logarithmic space), then } L \text{ can be decided by a Turing machine running in polynomial time (or logarithmic space) with an advice of size } O(f \log f).\]

**Proof.** Since $k$ is a constant and the alphabet is fixed, a $k$-tape Turing machine of size $f$ can be described with a string of length $O(f \log f)$. Then the result follows from Proposition 4.15.

To prove a converse result and simulate Turing machines with advice by sequences of Turing machines, we first prove that Property 4.2 holds.

**Proposition 4.17.** Let $M_1$ and $M_2$ be the class of Turing machines. Then, Property 4.2 holds, with the functions $m = m'$, $T = T'$ and $S = S'$.

**Proof.** Consider an arbitrary Turing machine $M_2$ of type $M_2$, an integer $n$ and a string $w$. Since $M_2$ is also of type $M_1$, the same machine can be used to simulate $M_2$. It follows that the machine $M_1$ is of the same size and runs in the same time and space.

**Proposition 4.18.** Let $f$ be an integer-valued function. If a language $L$ is decided by a Turing machine running in polynomial time (or logarithmic space) with an advice of size $f$, then $L$ can be decided by a sequence of one-tape Turing machines of size $O(f)$ running in polynomial time (or logarithmic space).

**Proof.** Let $M$ be a Turing machine machine of size $m'$ that decides $L$ in time $T'$ and space $S'$ with an advice of size $f$. It follows from Proposition 4.17 that Property 4.2 is satisfied with $m = m'$, $T = T'$ and $S = S'$. Turing machines satisfy Property 4.5 and 4.6, so by Theorem 4.13, the language $L$ can be decided by a sequence of Turing machines of size $O(m' + f)$, running in time $O(T)$ and space $O(S)$. These machines can be converted to Turing machines with one tape, using time $O(T^2)$ and space $O(S)$ (see Hartmanis and Stearns[16]).

However, the machine $M$ has both the input and the advice on its input tape, so $T'$ is a function of $n + f(n)$, while $T$ is a function of $n$. Thus, the running time is $O((T'(n + f(n)))^2)$. Since $T'$ and $f$ are polynomial, it follows that the running time of the sequence of machines is polynomial too. Similarly, the space usage of the sequence of machines is $O(S'(n + f(n)))$, which is logarithmic if $S'$ is logarithmic and $f$ is polynomial. Furthermore, $M$ is a single machine, so $m'$ is a constant. This implies that the sequence consists of Turing machines that are of size $O(f)$ and have a fixed number of tapes.
Corollary 4.16 and Proposition 4.18 enable us to give simple characterizations of the computational power of sequences of time- or space- bounded Turing machines in terms of advice classes.

**Corollary 4.19.** Let $k$ be an arbitrary positive integer. A language $L$ belongs to $P/poly$ iff $L$ can be decided by a sequence of polynomially sized $k$-tape Turing machines running in polynomial time.

**Corollary 4.20.** Let $k$ be an arbitrary positive integer. A language $L$ belongs to $LOGSPACE/poly$ iff $L$ can be decided by a sequence of polynomially sized $k$-tape Turing machines running in logarithmic space.

**Proof (for Corollary 4.19 and 4.20).** Observe that if an integer-valued function $m$ is a polynomial, then any function in $O(m \log m)$ is polynomially bounded. The results follow from Corollary 4.16 and Proposition 4.18.

4.3 Sequences of Machines with Bounded Description Sizes

If we compare Proposition 4.18 to Corollary 4.16, we see that the advice size blows up logarithmically. This is a consequence of Property 2.3, which states that an arbitrary machine of size $m$ needs a description length of at least $O(m \log m)$. However, the machines that we constructed in the proof of Theorem 4.13 are not arbitrary machines. We can use this fact to our advantage to find an encoding such that these particular machines can be described more efficiently.

**Proposition 4.21.** There is an encoding mechanism $e$ such that the sequence of machines of Theorem 4.13 have descriptions of size $O(f + m \log m)$.

**Proof.** We need to find an encoding such that $M_n$ can be described using a string of length $O(f + m \log m)$, for every $n$. Observe that $M_n$ consists basically of two parts, a part which stores $w = \alpha(n)$ and a part which calls $M_{n,w}$ on $w$ and the input. We encode $M_{n,w}$ with our original encoding. This requires a string of length $O(m \log m)$. To complete the description, we need a description of a machine that stores $w$ and calls $M_{n,w}$. This can be done with a string of length $O(|w|)$. Since $|w| = f(n)$, the machine $M_n$ can be described with a string of length $O(f + m \log m)$.

A general machine of type $M_2$ is encoded using $e$ as follows: if the machine consists of storing data and calling another machine, then we use the above description. Otherwise, we use the original encoding.

As an example, we can apply Proposition 4.21 to the case of Turing machine models with suitable complexity measures.

**Proposition 4.22.** Let $f$ be an integer-valued function. If a language $L$ is decided by a Turing machine $M$ running in polynomial time (or logarithmic space) with an advice of size $f$, then $L$ can be decided by a sequence of Turing machines running in polynomial time (or logarithmic space), with descriptions of size $O(f)$. 

Proof. Let $M$ have size $m'$. By Proposition 4.18, the sequence of Turing machines runs in polynomial time (or logarithmic space). By Proposition 4.21, the machines in the sequence have descriptions of size $O(f + m \log m)$. It follows from 4.17 that $m = m'$. As $m'$ is a constant, we conclude that the machines in the sequence have descriptions of size $O(f)$.

Corollary 4.23. Let $f$ be an integer-valued function. A language $L$ belongs to $P/O(f)$ iff $L$ can be decided by a sequence of Turing machines running in polynomial time, with descriptions of size $O(f)$.

Corollary 4.24. Let $f$ be an integer-valued function. A language $L$ belongs to $\text{LOGSPACE}/O(f)$ iff $L$ can be decided by a sequence of Turing machines running in logarithmic space, with descriptions of size $O(f)$.

Proof (of Corollary 4.23 and 4.24). The results follow from Proposition 4.15 and 4.22.

4.4 Sequences of Multi-Head Finite Automata

Multi-head finite automata are a common, powerful system model. We will show how the uniform framework for the equivalence result can be used to characterize the computational power of sequences of multi-head finite automata of various kinds. We will show the following result, based on familiar techniques from automata theory brought together in one framework.

Theorem 4.25. Let $k$ and $k'$ be positive integers. Then, the following statements about a language $L$ are equivalent:

(i) $L$ is in $\text{LOGSPACE}/\text{poly}$,
(ii) $L$ can be decided by a sequence of polynomially sized logarithmic space $k$-tape Turing machines,
(iii) $L$ can be decided by a sequence of polynomially sized deterministic finite automata with $k'$ heads,
(iv) $L$ can be decided by a finite automaton with one input head and an advice of polynomial size.

The equivalence between (i) and (ii) has already been shown in Corollary 4.20. Next, we will complete the proof by showing the equivalence between (i) and (iii) and the implications “(i) $\Rightarrow$ (iv)” and “(iv) $\Rightarrow$ (iii)”.

4.4.1 Turing Machines with Advice

Let $M_1$ be the model of finite automata and $M_2$ the model of Turing machines. To show that “(i) $\Rightarrow$ (iii)” of Theorem 4.25 holds, we use a common technique to simulate Turing machines on inputs of a fixed length with finite automata: Turing machines execute their program in a discrete fashion, thus the operation of such a
machine can be described by a sequence of configurations the machine is in from the initial stage to the moment the machine halts. See Balcázar et al.[1] for details on configurations. The simulation starts in the initial configuration and proceeds by moving from configuration to configuration, accepting iff the final configuration is an accepting configuration. This technique relies on the fact that the space usage of the Turing machine is bounded, so there are only finitely many configurations.

**Proposition 4.26.** If a language \( L \) is decided by a Turing machine running in logarithmic space, with an advice of polynomial size, then \( L \) can be decided by a sequence of polynomially sized two-way finite automata with one head.

**Proof.** A configuration of a Turing machine consists of the state the machine is in, the contents of the work-tapes and the position of the tape heads. Additionally, if the input head is on a symbol of the advice, the corresponding configuration also contains this symbol.

Let \( M \) be a Turing machine of size \( m' \) with \( k \) work tapes that decides \( L \) in space \( S' \), using an advice of size \( f \). Let \( c \) be the size of the tape-alphabet that \( M \) uses. In this case, the number of configurations for inputs of length \( n \) is \( O(m'^c S'(n+f(n))^k (n+f(n))) \). Since the advice is fixed for inputs of length \( n \), storing parts of the advice in the configurations does not increase the number of configurations. Thus, \( M \) can be simulated on inputs of length \( n \) by a finite automaton \( M_n \) of size \( m(n) \in O(m'^c S(n+f(n))^k (n+f(n))) \). Since \( S' \) is logarithmic, \( m \) is polynomial.

Let \( x \) be an input of length \( n \) and \( w \) the advice for \( x \). If the input head is on a symbol of \( x \) for a configuration, then \( M_n \) reads the corresponding symbol from its input tape. This is automatically the case in the initial configuration, the transition function ensures that this is always the case. The transition function of \( M_n \) mimics the transition function of the sequence of configurations (again, see Balcázar et al.[1] for details). There is a small catch, namely, the Turing machine has \( \langle x, w \rangle \) on its input tape, whereas the finite automaton has only \( x \). The transition function works around this by using the symbol of \( w \) that is stored in the configuration when the configuration tries to access a tape-square beyond the end of the tape. This completes the proof.

\( \Box \)

The next Proposition shows that “(iii) \( \Rightarrow \) (i)” of Theorem 4.25 holds.

**Proposition 4.27.** If a language \( L \) is decided by a sequence of polynomially sized two-way deterministic finite automata with \( k' \) heads, then \( L \) can be decided by a Turing machine running in logarithmic space, with an advice of polynomial size.

**Proof.** A finite automaton can be viewed as a Turing machine without work-tapes. So, if we use the standard encoding for Turing machines to describe finite automata, then a universal Turing machine \( M \) satisfies Property 4.1. For a polynomially sized Turing machine, the description has polynomial size, thus \( M \) uses an advice of polynomial size. Let \( m'(n) \) be the size of the \( n \)-th automaton of the sequence. For a Turing machine without work-tapes, only the state and the positions of the \( k' \) heads need to be stored, so \( M \) uses space \( S \in O(\log m'(n) + k' \log n) \). Since \( m' \) is polynomial and \( k' \) is fixed, \( S \) is logarithmic in the input.

\( \Box \)
An interesting consequence is that the number of tape heads used in a sequence of polynomially sized finite automata has little impact on the computational power of the sequence.

**Corollary 4.28.** For any integer \( k \), sequences of polynomially sized two-way deterministic finite automata with \( k \) heads can be simulated by sequences of polynomially sized two-way deterministic finite automata with one head.

**Proof.** This follows directly from Propositions 4.27 and 4.26. \( \square \)

Let \( L \) be a language over an alphabet \( \Sigma \). Suppose a Turing machine decides \( L \) using linear space and an exponentially bounded advice. It follows from the proof of Proposition 4.26 that \( L \) can be decided by a sequence of exponentially sized finite automata. However, in such a case, it may be more efficient to decide the finite set \( L \cap \Sigma^n \) directly, without simulating the Turing machine. This can be done by a one-way finite automaton with \( |\Sigma|^n \) states even for \( |\Sigma| = 1 \). In fact, this can be done with any language.

**Theorem 4.29.** Any language can be decided by a sequence of exponentially sized one-way deterministic finite automata.

Note that for sparse languages, polynomially sized automata will do.

### 4.4.2 Multi-Head Finite Automata with Advice

When we have the freedom to choose the descriptions, we can add all sorts of helpful information in the description to help with the simulation of the described machines. This allows us to simulate sequences machines with less powerful machines that were unable to simulate the machines of the sequence using the standard encoding. We illustrate this with finite automata with advice.

Using finite automata in a sequence of machines is very straightforward. Using a finite automaton to simulate a sequence of machines on the other hand, is somewhat more complicated. A finite automaton with advice is a two-way finite automaton with two heads for its input tape. Alternatively, one may view a finite automaton with advice as an automaton with two tapes, one containing the input, the other containing the advice for the length of the input. The two heads are necessary, since the automaton cannot remember the position of a head on the tape, so it cannot move from input to advice and back using just one head. More generally, on a multi-head finite automaton with advice, one of the heads is designated as the *advice head*, the other heads are regular *input heads*. We assume that the advice is stored on the same tape as the input, as it simplifies the constructions somewhat.

Given a description of an automaton, a simulating automaton needs to move from state to state on this description. The automaton can only use its finite control to determine if it has found the correct state on the description. When the number of states of the simulated machine is unbounded, the simulating machine will run out of states to test this. Thus, in general, a finite automaton with advice cannot simulate a sequence of automata.
4.4 Sequences of Multi-Head Finite Automata

The argument suggests that if there is a description for automata such that the possible destination states of any transition can be located with a finite amount of information, then these automata can be simulated with a finite automaton. We will show that this is indeed the case.

Property 4.30 captures the conditions a sequence of finite automata needs to satisfy in order to be simulated by a finite automaton with advice.

**Property 4.30.** Consider a sequence of finite automata with \(k\) heads, for some integer \(k\). There is an encoding mechanism \(e\) for the automata in the sequence and finite automata \(M_{\text{init}}\) and \(M_{\text{step}}\) with \(k+1\) heads such that for an automaton \(M_n\) in the sequence, the description of \(M_n\) using \(e\) has the following properties:

- The description is based on a transition-list,
- the initial state of \(M_n\) can be found by \(M_{\text{init}}\) if its input tape contains the description of \(M_n\),
- for each state \(q\) of \(M_n\), every input \(x\) of \(M_n\), and every possible placement of the \(k\) heads on \(x\), which cause the transition function to output state \(q'\), if the finite automaton \(M_{\text{step}}\) contains \(x\), as well as the description of \(M_n\) on its input tape, a tape head positioned at the beginning of \(q\) on this description and the other \(k\) heads placed at the positions of the input, then \(M_{\text{step}}\) moves a tape head to the beginning of \(q'\) and the remaining \(k\) heads to the positions corresponding to the transition.

Now, take \(\mathcal{M}_1\) as the class of finite automata satisfying Property 4.30 and \(\mathcal{M}_2\) as the class of finite automata. Proposition 4.31 shows that a sequence of automata that satisfies Property 4.30 for an encoding mechanism \(e\) satisfies Property 4.1.

**Proposition 4.31.** Consider a sequence of finite automata with \(k\) heads running in time \(T'\) with descriptions of size \(f\), using an encoding mechanism \(e\) that satisfies Property 4.30. This sequence satisfies Property 4.1 for this encoding mechanism, with \(T \in O(T' \cdot f)\) and \(S = 0\). Furthermore, if the automata \(M_{\text{init}}\) and \(M_{\text{step}}\) are of size \(m_{\text{init}}\) and \(m_{\text{step}}\) respectively, then the finite automaton \(M\) is of size \(O(m_{\text{init}} + m_{\text{step}})\).

**Proof.** Let \(M_1, M_2, \ldots\) be the finite automata in the sequence. We construct a finite automaton \(M\) with \(k+1\) heads and an advice mechanism that simulates the sequence. For inputs of length \(n\), the advice contains the description of \(M_n\). The advice head is the \((k+1)\)-st head, so \(M\) may call the automata from Property 4.30.

The simulating automaton \(M\) starts by moving the \(k\) input heads to the first symbol of the input. This way, the head positions of \(M\) correspond to the head positions of \(M_n\) before the first transition occurs. Now, \(M\) calls \(M_{\text{init}}\) to move the remaining head to the beginning of the initial configuration. This can be done in \(O(f)\) steps. After this, \(M\) repeatedly calls \(M_{\text{step}}\) to find the next state of the computation, moving the input heads to the correct positions along the way, until a final state is reached. Each call takes at most \(O(f)\) time.

Since finite automata satisfy Property 4.6, \(M\) is of size \(O(m_{\text{init}} + T' \cdot m_{\text{step}})\). However, this direct construction does not yield a finite automaton since \(T'\) is not a constant. Instead, after each call of \(M_{\text{step}}\), the finite automaton \(M\) checks if the
state $M_n$ is in is a final state. This can be done with $O(1)$ states. But then, $M$ can just use the same copy of $M_{\text{step}}$ in its finite control over and over again. Thus, $M$ is of size $O(m_{\text{init}} + m_{\text{step}})$.

The finite automaton $M$ simulates $M_n$ using $O(T' \cdot f)$ time. Note that finite automata have no implementation of mutable data, so $S = 0$. Thus, Property 4.1 is satisfied with $T \in O(T' \cdot f)$ and $S = 0$.

By combining the results in the subsection, we obtain the following result.

**Proposition 4.32.** Let $L$ be a language decided by a sequence of finite automata with $k$ heads and descriptions of size $f$ satisfying Property 4.30. Then $L$ can be decided by a finite automaton with $k$ input heads and an advice of size $f$.

**Proof.** The result follows from Proposition 4.31 and Theorem 4.12. □

**Sequences of Finite Automata with Bounded Bandwidth**

As an example, we apply Proposition 4.31 to sequences of finite automata with bounded bandwidth. A matrix has bandwidth $b$ if $b$ is the smallest integer for which the entries with index $(i, j)$ are empty whenever $|j - i| \geq b$. A finite automaton has bandwidth $b$ if there is an ordering of the states such that the transition matrix for this ordering has bandwidth $b$.

**Example 4.33.** Consider a sequence of finite automata with $k$ heads and bandwidth bounded by $b$, operating over an alphabet of size $c$. We construct a description of the automata in the sequence that satisfies Property 4.30. We order and list the states of an automaton such that the transition matrix has bandwidth $b$, starting with the initial state. From any state in this list, the next states all lie within the $b$ previous and $b$ next states. Thus, we describe the next state with an integer $i$ between $-b$ and $b$.

An automaton of size $m$ has $mc^k$ transitions and each transition needs a string of length $\log c^k + \log b$. It follows that an automaton with $m$ states can be described with a string of length $O(mc^k \cdot \log(c^k b))$.

Since the first state in the list is the initial state, the finite automaton $M_{\text{init}}$ halts immediately, using just one state.

To determine the next transition, the finite automaton $M_{\text{step}}$ needs the symbols under the heads. For an automaton with $k$ heads, $O(k)$ states are needed to find the transition corresponding to the $k$ symbols under the heads. The head movements, which are listed next, can be applied with $O(1)$ states. The next state is determined by parsing the integer $i$ belonging to this transition. This can be done with a full tree of size $O(b)$.

Now, if we place distinguishing markers between every state and its transitions, then we can move in the list in the proper direction until we encounter the $i$-th marker to find the next state. This can be done with another $i$ states. To sum up, $M_{\text{step}}$ uses $O(k + b^2)$ states.
Proposition 4.34. If a language $L$ over an alphabet of size $c$ can be decided by a sequence of automata of size $f$ with $k$ heads and bandwidth bounded by an integer $b$, then $L$ can be decided by a finite automaton $M$ of size $O(k + b^2)$ with $k$ input heads and an advice of length $O(f(n)c^k \cdot \log(c^kb))$.

Proof. We use the description and the automata $M_{\text{init}}$ and $M_{\text{step}}$ from Example 4.33. By Proposition 4.31, the automaton $M$ is of size $O(k + b^2)$. By Proposition 4.32, the automaton $M$ decides $L$ using the descriptions of the automata in the sequence as advice.

Observe that we don’t use the descriptions of the states, we only need the descriptions of the transitions and the markers. Thus, the length of the description of the $n$-th automaton in the sequence is $O(f(n)c^k \cdot \log(c^kb))$.

Since $c$, $k$ and $b$ are constants, it follows that a sequence of $k$-head finite automata of size $f$ and bandwidth bounded by an integer $b$ can be simulated by a finite automaton with $k$ input heads and an advice of size $O(f)$.

Corollary 4.35. If a language $L$ can be decided by a sequence of $k$-head polynomially sized finite automata with bandwidth bounded by an integer $b$, then $L$ can be decided by a finite automaton with $k$ input heads and a polynomially sized advice.

Corollary 4.36. If a language $L$ can be decided by a sequence of $k$-head logarithmically sized finite automata with bandwidth bounded by an integer $b$, then $L$ can be decided by a finite automaton with $k$ input heads and a logarithmically sized advice.

Log-Space Turing Machines with Advice

As another example, consider the sequence of finite automata that is constructed in Proposition 4.26. This sequence satisfies Property 4.30. We use this fact to show that “$(i) \Rightarrow (iv)$” of Theorem 4.25 holds.

Proposition 4.37. Let $L$ be a language in LOGSPACE/poly. Then, $L$ can be decided by a sequence of finite automata with one head and polynomially sized descriptions that satisfy Property 4.30.

Proof. Let $L$ be decided by a Turing machine $M$ in logarithmic space, using an advice of polynomial size. By Proposition 4.26, $L$ can be decided by a sequence of finite automata. We modify the automata in the sequence, by adding a few extra states to the automata, to simplify the order of the states. Since the transition function remains unaltered, this new sequence decides the same language $L$. We construct a description for this sequence of automata that satisfies Property 4.30.

Let $m'$ be the number of states of the Turing machine $M$. Let $M_n$ be the $n$-th automaton in the sequence. A state of $M_n$ corresponds to a tuple $(q, v_1, \ldots, v_k, l, a_l)$, where $q$ is a state of $M$, $l$ is the unary encoding of the position of the advice head of $M$, $a_l$ is the symbol under the advice head and $v_i$ is a string representing the contents of the $i$-th work tape of $M$. 
The string $v_i$ consists of symbols from the input alphabet plus the blank symbol and the extra symbol $\#$, which denotes the position of the head. Since the Turing machine $M$ works in logarithmic space, there is an integer $d$ such that the work tapes use at most $d \log n$ cells. Thus, the tape contents of $M$ fit into strings of length $d \log n$. For every $1 \leq i \leq k$, the string $v_i$ has length $d \log n$ (shorter tape contents get padded with blank symbols). A string $v_i$ is valid if it satisfies the following constraints:

- the string contains exactly one $\#$ symbol,
- the string does not have any blank symbols before a $\#$ symbol,
- the string does not have any non-blank symbols behind a blank symbol.

The symbol directly to the right of the $\#$ symbol corresponds to the symbol under the head of the $i$-th tape of $M$.

A state is valid iff all the strings in the tuple are valid. Note that a state is valid iff it corresponds to a configuration of $M$. Thus, the states that are added to the original sequence are exactly the states that are not valid.

Assume that the symbols of the strings have a numerical value and that the strings are ordered reversed lexicographically using this numerical value, i.e., the first symbol of a string is the least significant and the last symbol is the most significant. The description of $M_n$ lists all the states of $M_n$. The order of the states depends on $q, v_1, \ldots, v_k$ and $l$. These components are ordered reversed lexicographically ($l$ is the most significant bit and $q$ is the least significant bit). The numerical values are chosen such that the initial state of $M_n$ is the first state in the order.

Each state is accompanied by a list of transitions. Thus, each state corresponds to a labeled list of transitions. A marker is placed between every labeled list of transitions and another marker is placed between every transition within the labeled lists. Using these markers, a finite automaton can move a tape head between transitions or between states by simply counting the markers.

In addition to these labeled lists, the description starts with strings (yardsticks) of length $(c+2)^j + (i-1)d \log n$ for every $1 \leq i \leq k$ and every $0 \leq j \leq d \log n$, ordered lexicographically ($i$ is the most significant bit), separated by a third type of markers. A special marker is used to separate the yardsticks from the labeled lists.

Since $m'$ is constant, the length of the string $v_i$ is logarithmically bounded and $l$ is polynomially bounded, each state of $M_n$ can be described with string of polynomial length and $M_n$ contains polynomially many states. Since $i$ is finite and $j$ is logarithmically bounded, there are polynomially many yardsticks of polynomial length in the description. Thus, the description of $M_n$ is polynomial in $n$.

A finite automaton $M_{\text{init}}$ can find the initial state of $M_n$ by moving a tape head over the description of $M_n$ until the special marker is reached, using $O(1)$ states to recognize the marker.

Consider a string $v_i$, with $1 \leq i \leq k$. It is of the form

$$u_1 \ldots u_{j-2} u_{j-1} \# u_{j+1} u_{j+2} \ldots u_{d \log n} \ . \quad (4.1)$$

After a transition in which the $i$-th head of $M$ does not move, the next string $v'_i$ has the form of the second string in (4.2), where $a$ is a symbol from the alphabet.
\[ u_1 \ldots u_{j-2} \ u_{j-1} \]\[ u_1 \ldots u_{j-2} \ u_{j-1} \]\[ u_{j+1} \ u_{j+2} \ldots u_{d \log n} \]\[ a \ u_{j+2} \ldots u_{d \log n} \] \quad (4.2)

It follows from (4.2) that the two strings differ only in one place. In the reversed lexicographical order, the string \( v_i' \) is located \((a - u_{j+1})(c+2)^j\) strings to the right from \( v_i \).

If the head moves right during the transition, the string \( v_i' \) has the form of the last string in (4.3). Notice that the intermediate strings differ only in one place.

\[ u_1 \ldots u_{j-2} \ u_{j-1} \]\[ u_1 \ldots u_{j-2} \ u_{j-1} \]\[ u_{j+1} \ u_{j+2} \ldots u_{d \log n} \]\[ a \ u_{j+2} \ldots u_{d \log n} \] \quad (4.3)

The string \( v_i' \) is located \((a - u_{j+1})(c+2)^j\) plus \((a - \sharp)(c+2)^{j-1}\) strings to the right of \( v_i \).

If the head moves left during the transition, the next string \( v_i' \) has the form of the last string in (4.4).

\[ u_1 \ldots u_{j-2} \ u_{j-1} \]\[ u_1 \ldots u_{j-2} \ u_{j-1} \]\[ u_{j+1} \ u_{j+2} \ldots u_{d \log n} \]\[ a \ u_{j+2} \ldots u_{d \log n} \] \quad (4.4)

The string \( v_i' \) is located \((a - u_{j+1})(c+2)^j\) plus \((\sharp - u_{j-1})(c+2)^{j-2}\) plus \((u_{j-1} - \sharp)(c+2)^{j-1}\) strings to the right of \( v_i \).

Suppose the first head of \( M_{\text{step}} \) is located on the description of a state \((q, v_1, \ldots, v_k, l, a_l)\). A transition is simulated by moving the head to the description of the next state. The description of the transition includes the symbols under the \( k \) heads, the \( k \) head movements, the next state \( q' \) of \( M \) and the next values of \( l' \) and \( a_{l'} \). The location of the next state is found by first moving to the state \((q', v_1, \ldots, v_k, l, a_l)\). Note that this state lies at most \( m \) states to the left or right of the original state. Thus, it can be found by a finite automaton.

Then, the contents of \( v_i \) are replaced by \( v_i' \), one by one, until the first head of \( M_{\text{step}} \) is on the description of the state \((q', v'_1, \ldots, v'_k, l, a_l)\). In the case of (4.2), this requires moving \((a - u_{j+1})(c+2)^j\) times \( m' \cdot (c+2)^{(i-1)d \log n} \) states to the right to replace \( v_i \). Since \((a - u_{j+1})\) and \( m' \) are bounded and the description contains a yardstick of length \((c+2)^{j+(i-1)d \log n}\), this move can be done by a finite automaton. However, this yardstick has to be located by \( M_{\text{step}} \) first. Note that this yardstick is the \((j + (i-1)d \log n)\)-th yardstick in the list. Thus, \( M_{\text{step}} \) can move the second head to this yardstick, using \( i \) (in its finite control), \( d \log n \) (the length of the string \( v_i \)) and \( j \) (indicated by the \( \sharp \) symbol). Then, \( M_{\text{step}} \) can use the second head and the yardstick to move the first head to the next state. The situations for (4.3) and (4.4) are handled similarly. (Note that the order of the yardsticks implies that the yardsticks of length \((c+2)^{j-2+(i-1)d \log n}\), \((c+2)^{j-1+(i-1)d \log n}\) and \((c+2)^{j+(i-1)d \log n}\) are right next to each other.)

Finally, \( l \) and \( a_l \) need to be updated. The value of \( a_{l'} \) is determined by \( l' \), which is between \( l - 1 \) and \( l + 1 \). Thus, the state \((q', v'_1, \ldots, v'_k, l', a_{l'})\) is located another \( m' \cdot (c+2)^{k \cdot d \log n} \) states to the left or right. This move can be done in a similar fashion as well.
There is a small problem with this method. Both tape heads are used to find the next state. Thus, we lose the position of the input head of $M$. Fortunately, $l$ encodes the position of the input head of the Turing machine when $l \leq n$, so the second head can be moved back to the correct input position, using the encoding of $l$. If $l > n$, then the input is not used by the Turing machine, instead, an advice symbol is read. In this case, the second head should just move to the last input symbol. It follows that $M_{\text{step}}$ can move from state to state on a tape containing this description, updating the heads after each move. Thus, the description satisfies Property 4.30.

\[\square\]

**Proposition 4.38.** Let $L$ be a language decided by a Turing machine running in logarithmic space with an advice of polynomial size. Then $L$ can be decided by a two-way finite automaton with one input head and an advice of polynomial size.

**Proof.** The result follows from Propositions 4.37 and 4.32.

\[\square\]

It is already known that polynomially sized finite automata with multiple heads have the same computational power as Turing machines running in logarithmic space (see Wagner and Wechsung[44]). The number of heads of the automaton depends on the constant factor of the logarithm. We see however, that when both devices may use an advice mechanism, one input head (plus one head for the advice) for the finite automaton is sufficient, independent of the factor of the logarithm.

**Multi-Head Finite Automata with Advice**

Next, we show that “(iv) $\Rightarrow$ (iii)” of Theorem 4.25 holds. It is a direct consequence of the stronger statement in Proposition 4.39.

**Proposition 4.39.** Let $L$ be a language over an alphabet of size $c$ decided by a finite automaton of size $m'$ with $k$ input heads and an advice of length $f$. Then $L$ can be decided by a sequence of finite automata of size $m'f(n)$ with $k$ heads and descriptions of size $O(m'f(n)c^k \log(c^k m'))$ satisfying Property 4.30.

**Proof.** The proof consists of two parts. The first part shows that an arbitrary finite automaton with advice can be simulated by a sequence of finite automata. The second part shows that the particular sequence that was constructed in the first part satisfies Property 4.30, by giving a description that satisfies it.

Let $M$ be a finite automaton of size $m'$ with $k$ input heads and an advice of size $f$ that decides $L$. A configuration of $M$ consists of a state, the contents of the advice tape and the position of the advice head. For the $n$-th automaton $M_n$ in the sequence, its states are those configurations where the advice tape contains the $n$-th advice, and the (relative) position of the advice head is at most $f(n)$. Since the advice is fixed for inputs of length $n$, the $n$-th automaton has $m'f(n)$ states, tuples of the form $(q, w_j, j)$, where $q$ is a state from $M$ and $w_j$ is the $j$-th symbol of the $n$-th advice string, for $1 \leq j \leq f(n)$.
Now, \( M_n \) starts in the configuration that contains the initial state of \( M \) and the advice for inputs of length \( n \) and has the \( k \) tape heads in the first position. Then, \( M_n \) reads the symbols under the \( k \) heads. Using the transition function, the \( k \) heads of \( M_n \) simulate one move of the input heads of \( M \). Then, \( M_n \) changes to the configuration corresponding to the next state and the next position of the advice head of \( M \). This way, the sequence of finite automata decides \( L \).

Next, we give a description for the automata in the sequence that satisfies Property 4.30. We place the initial configuration first in the list, so that it can be found by a finite automaton \( M_{\text{init}} \).

Suppose that the automaton \( M_n \) of the sequence is in a state \( q \), with the advice head at position \( j \). Since the heads of \( M_n \) can move at most one position at a time, after one transition \( M_n \) is in a state \( q' \), with the advice positioned at \( j' \), such that \( j - 1 \leq j' \leq j + 1 \). Thus, if we order the states reversed lexicographically, then the transition matrix has a bandwidth of \( 2m' \). It follows from Example 4.33 that \( M_n \) has a description of size \( O(m' f(n)c^k \log(c^k m')) \) that satisfies Property 4.30.

\[ \square \]

**Corollary 4.40.** Let \( k \) be a positive integer. Let \( L \) be a language decided by a finite automaton with one input head and an advice of polynomial size. Then \( L \) can be decided by a sequence of polynomially sized finite automata with \( k \) heads.

**Proof.** This follows directly from Proposition 4.39 and the fact that \( m' \) is a constant.

\[ \square \]

Just as in Section 4.3, we can compare advice lengths to description sizes of finite automata. We combine the results in Theorem 4.41.

**Theorem 4.41.** A language \( L \) can be decided by a sequence of finite automata with \( k \) heads and descriptions of size \( O(f) \) satisfying Property 4.30 iff \( L \) can be decided by a finite automaton with \( k \) input heads and an advice of size \( O(f) \).

**Proof.** The result follows from Proposition 4.32, Proposition 4.39 and the fact that \( k, c \) and \( m' \) are constants.

\[ \square \]

### 4.5 Sequences of Multi-Head Pushdown Automata

As a next case, we consider multi-head automata with the simplest kind of external memory commonly considered in automata models, i.e., the pushdown store. We will show the following characterization of the computational power of sequences of multi-head pushdown automata, as another example of the power of the framework.

**Theorem 4.42.** Let \( k \) be a positive integer. The following statements about a language \( L \) are equivalent:

(i) \( L \) is in \( P/poly \),

(ii) \( L \) can be decided by a sequence of polynomially sized Boolean circuits,
(iii) \( L \) can be decided by a sequence of polynomially sized polynomial time \( k \)-tape Turing machines,

(iv) \( L \) can be decided by a sequence of polynomially sized two-way pushdown automata with \( k' \) heads, for a certain integer \( k' \),

(v) \( L \) can be decided by a pushdown automaton with \( k' \) input heads and an advice of polynomial size, for a certain integer \( k' \).

The equivalence between (i) and (ii) has already been shown by Karp and Lipton[22] (see Corollary 4.14). Corollary 4.19 shows the equivalence between (i) and (iii). We will complete the proof by showing the equivalence between (i) and (iv) and the equivalence between (iv) and (v). See also Petersen[37] for a direct approach to the equivalence between (i) and and (v).

4.5.1 Turing Machines with Advice

The next Proposition shows that “(i) \( \implies \) (iv)” of Theorem 4.42 holds. We will build on the ideas from Cook[9].

Proposition 4.43. If a language \( L \) is decided by a Turing machine running in polynomial time with an advice of polynomial size, then there is an integer \( k \) such that \( L \) can be decided by a sequence of polynomially sized two-way pushdown automata with \( k \) heads.

Proof. Consider a Turing machine that decides \( L \) in polynomial time \( T \), using an advice of polynomial size. Then, there is a Turing machine with one writable tape that decides \( L \), in time \( O((T(n))^2) \), which is polynomial too. This Turing machine moves its head in a sweeping pattern. By the results proved by Cook[9], this Turing machine can be simulated by a pushdown automaton with an additional work tape of length \( O(\log T(n)) \). Since \( T \) is polynomial, the work tape is logarithmically bounded. The finite control incorporates the transition function of \( M \), (using \( O(1) \) size), and the function of \( t \) which produces the position the head is in at time \( t \) (using \( O(1) \) size and a logarithmically bounded work-tape).

Now, this pushdown automaton is not the one we want, since it uses a work-tape and takes an advice with the input. We solve the latter by storing the advice directly in the finite control of the pushdown automaton (adding polynomially many states), and the former by noting that an automaton with logarithmically bounded work-tapes is equivalent to an automaton with \( k \) tape heads, for a certain integer \( k \), see Wagner and Wechsung[44]. Note that \( k \) depends on the running time of the Turing machine, so we can use the same \( k \) for all pushdown automata in the sequence. Putting everything together, the sequence uses polynomially sized pushdown automata with \( k \) heads to decide \( L \).

\[ \square \]

The following Proposition shows that “(iv) \( \implies \) (i)” of Theorem 4.42 holds.

Proposition 4.44. Let \( k \) be a positive integer. If a language \( L \) is decided by a sequence of polynomially sized two-way pushdown automata with \( k \) heads, then \( L \) can be decided by a Turing machine running in polynomial time with an advice of polynomial size.
Proof. The Turing machine takes the description of the pushdown automaton as advice. Polynomially sized pushdown automata have descriptions of polynomial size.

The simulation is achieved by examining realizable pairs of configurations of the pushdown automaton. A configuration consists of a state, the contents of the $k$ heads and the symbol on top of the pushdown store. For the definition of realizable pairs and the operation to yield a new realizable pair given two realizable pairs, see Cook[9]. A realizable pair $(C, C')$ is accepting if $C$ is the initial configuration and $C'$ is an accepting configuration. The Turing machine starts by storing all realizable pairs $(C, C)$. Then, the machine repeatedly applies the yield operation to all pairs in this list, adding the newly found pairs to this list, until an accepting pair has been found (in which case the machine accepts) or no new realizable pairs can be found anymore (the input is rejected).

To perform the yield operation, the Turing machine has to look up the transition function from its advice and apply it to the configurations in the pairs. Let $f(n)$ be the size of the description of the $n$-th pushdown automaton $M_n$. Then an application of the yield operation takes $O(f)$ time.

Let $c$ the size of the alphabet of the pushdown automata. If $M_n$ has $m(n)$ states, then there are $m(n)n^k c$ configurations. To avoid confusion, we refer to realizable pairs as items in the list. Let $r_i$ be the number of items in the list after the yield operation has been applied to all pairs of items in the list $i$ times. Then $r_0 = m(n)n^k c$. The yield operation has to be applied to all pairs of items in the list, so the $i$-th round takes $O((r_i)^2 f)$ time. In the worst case, $r_i$ increases by one each round and every pair of configurations is realizable. Then, the time to compute all realizable pairs is

$$
\sum_{j=r_0}^{(r_0)^2} O (j^2 f) = O \left( (r_0)^4 f \right) = O \left( (m(n)n^k c)^4 f(n) \right).
$$

Thus, all realizable pairs can be found in polynomial time.

\[ \square \]

4.5.2 Multi-Head Pushdown Automata with Advice

The main difference between a finite automaton and a pushdown automaton is that a pushdown automaton can store infinite amounts of information in a pushdown store. The pushdown store is stored on a separate tape with one tape head. We define a pushdown automaton with advice capabilities as a pushdown automaton with two heads for its input tape (or two tapes, one containing the input, the other containing the advice).

Ibarra[20] proved that we cannot simulate a pushdown automaton with multiple heads by a pushdown automaton with one head. It follows that the advice mechanism requires at least two heads. On a multi-head pushdown automaton with advice, one of the heads of the input tape is designated as the advice head, the other heads are the input heads.

We list a property of descriptions of sequences of pushdown automata that enables us to simulate the sequence with a pushdown automaton that takes these descriptions as advice.
Property 4.45. Let \( k \) be an integer. Consider a sequence of pushdown automata with \( k \) heads. There is an encoding mechanism for the automata in the sequence and pushdown automata \( M_{\text{init}} \) and \( M_{\text{step}} \) with \( k + 1 \) heads such that for an automaton \( M \) in the sequence, the description of \( M \) using this mechanism has the following properties:

- The description is based on a transition-list,
- the initial state of \( M \) can be found by \( M_{\text{init}} \) if its input tape contains the description of \( M \),
- for each state \( q \) of \( M \), every input \( x \) of \( M \), every possible placement of the \( k \) heads on \( x \) and every string \( y \) in the pushdown store of \( M \), which cause the transition function to output state \( q' \) and replace \( y \) by \( y' \) in the pushdown store, if the pushdown automaton \( M_{\text{step}} \) contains \( x \), as well as the description of \( M \) on its input tape, a tape head positioned at the beginning of \( q \) on this description, the other \( k \) heads placed at the same positions of the input and the pushdown store contains \( y \), then \( M_{\text{step}} \) moves a tape head to the beginning of \( q' \) and the remaining \( k \) heads to the positions corresponding to the transition and replaces \( y \) by \( y' \) in the pushdown store.

Note that Property 4.45 is very similar to Property 4.30, only the occurrences of the word “finite” are replaced by the word “pushdown” and the contents of the pushdown store are taken into account. Another difference between the two Properties is that it is unlikely that every finite automaton has a description satisfying Property 4.30, while there is an encoding that satisfies Property 4.45 for every sequence of pushdown automata.

Proposition 4.46. Consider a sequence of two-way pushdown automata with \( k \) heads, for an integer \( k \). If the automata are described by transition-lists, then the descriptions satisfy Property 4.45.

Proof. Since the initial state is marked, the pushdown automaton \( M_{\text{init}} \) can move an input head on the input tape holding the description of \( M \) until it locates this (finite) marker, using \( O(1) \) states.

The pushdown automaton \( M_{\text{step}} \) contains the input to \( M \) as well as the description of \( M \) on its input tape. One head is positioned at the beginning of the description of a state \( q \), the remaining \( k \) heads are positioned on the input cells corresponding to the positions of the \( k \) heads of \( M \). The pushdown store of \( M_{\text{step}} \) contains the pushdown store contents of \( M \).

The automata all operate over a fixed alphabet, say of size \( c \). Then, the number of transitions starting in one state is at most \( c^{k+1} \). Thus, we need \( O(c^k) \) states to distinguish between the different descriptions of transitions following \( q \).

Suppose the transition corresponding to the \( c^{k+1} \) tape symbols is found. If the transition pops a symbol from the pushdown store of \( M \), then \( M_{\text{step}} \) pops its pushdown store too. Similarly, if a symbol is pushed onto the pushdown store of \( M \), the same symbol is pushed onto the pushdown store of \( M_{\text{step}} \). Then, \( k \) of the heads of \( M_{\text{step}} \) are moved according to the transition of \( M \).

Next, a special marker is pushed onto the pushdown store, followed by the description of \( q' \) (using \( O(1) \) states). The remaining head is moved to the end of the description and from there \( M_{\text{step}} \) searches for a match to the state in the
pushdown store. This is done by using the markers between the states in the list to find the end of a state. From here, the head moves backwards until the previous marker is reached, comparing the symbol under the pointer head with the symbol that is popped from the pushdown store. If the symbols don’t match, or if either one reaches the end before the other, then the head is moved forwards again, pushing back the symbols that were popped off the pushdown store. Then, the head is moved backwards to the end of the state before the one it is on now and tries again. Eventually, \( q' \) will be found. Now, the marker is popped from the pushdown store again, so that \( M_{\text{step}} \) contains the correct contents of the pushdown store again. The number of states used in this part of the procedure depends only on the size of the markers. Thus, \( O(1) \) states are sufficient.

All in all, \( M_{\text{step}} \) uses finitely many states to move the heads to the correct positions and modify the pushdown store.

The implication \( (iv) \Rightarrow (v) \) of Theorem 4.42 is a direct consequence of the next Proposition. It is stated in Corollary 4.48.

**Proposition 4.47.** Let \( L \) be a language that is decided by a sequence of pushdown automata with \( k \) heads running in time \( T \) with descriptions of size \( f \), using an encoding that satisfies Property 4.45 for a certain integer \( m \). Then, \( L \) can be decided by a pushdown automaton with \( k \) input heads and an advice of size \( f \).

**Proof.** Let \( M_1, M_2, \ldots \) be the pushdown automata in a sequence that decides \( L \). We construct a pushdown automaton \( M \) with advice that decides \( L \). For inputs of length \( n \), the advice contains the description of the \( n \)-th pushdown automaton \( M_n \). The advice head is used as the \( (k+1) \)-st tape head, so \( M \) may call the pushdown automata described in Property 4.45.

The simulating pushdown automaton \( M \) starts by moving the \( k \) heads to the first symbol of the input. Now, \( M \) calls \( M_{\text{init}} \) to find a pointer (the advice head) to the initial state. This takes \( O(1) \) states. After this, \( M \) can repeatedly call \( M_{\text{step}} \) to find the next state of the computation, updating the head positions and the pushdown store along the way, using \( O(1) \) states, until a final state is found. It follows that \( M \) simulates \( M_n \) using a finite number of states.

**Corollary 4.48.** Let \( L \) be a language that is decided by a sequence of polynomially sized pushdown automata with \( k \) heads. Then \( L \) can be decided by a pushdown automaton with \( k \) input heads and an advice of polynomial size.

**Proof.** If a pushdown automaton has polynomial size, then its transition-list has polynomial size. The result then follows from Propositions 4.46 and 4.47.

Corollary 4.50 proves that \( (v) \Rightarrow (iv) \) of Theorem 4.42 holds. It follows from the next Proposition.

**Proposition 4.49.** Let \( L \) be a language that is decided by a pushdown automaton of size \( m' \) with \( k \) heads and an advice of length \( f \). Then \( L \) can be decided by a sequence of pushdown automata of size \( O(m'f(n)) \) with \( k \) heads.
Proof. Let $M$ be a pushdown automaton of size $m'$ with advice of size $f$ that decides $L$. We are going to simulate $M$ by moving from configuration to configuration again. Since the pushdown store can be unbounded, the number of configurations is unbounded too, so we exclude the contents of the pushdown store from the configurations. Instead, the contents of the pushdown store are stored in the pushdown store of the machines in the sequence. Thus, a configuration consists of a state, the contents of the advice tape and the position of the advice head. For the $n$-th pushdown automaton in the sequence, the advice tape contains the advice for strings of length $n$ and the (relative) position of the advice head is at most $f(n)$. So, the $n$-th pushdown automaton has $O(m'f(n))$ states.

The simulation proceeds similar to previous results (e.g. Proposition 4.39), except the symbol at the top of the pushdown store is also taken into account to determine the next configuration. Of course, the pushdown store is synchronized with the pushdown store of $M$ at every step.

\[ \square \]

Corollary 4.50. Let $k$ be a positive integer. Let $L$ be a language decided by a pushdown automaton with $k$ input heads and an advice of polynomial size. Then $L$ can be decided by a sequence of polynomially sized pushdown automata with $k$ heads.

Proof. This follows from Proposition 4.49 and the fact that $m'$ is a constant.

\[ \square \]

We finish this section by comparing advice lengths to description sizes of pushdown automata.

Theorem 4.51. A language $L$ can be decided by a sequence of pushdown automata with $k$ heads and descriptions of size $O(f)$ based on transition-lists iff $L$ can be decided by a pushdown automaton with $k$ input heads and an advice of size $O(f)$.

Proof. The first part of the equivalence follows from Propositions 4.46 and 4.47. The sequence of pushdown automata constructed in Proposition 4.49 has a bounded bandwidth, so the machines in the sequence can be described as in Example 4.33. Since $c$, $k$ and $m'$ are constants, the result follows.

\[ \square \]

4.6 Using Transition Lists or Flow Lists for Advice

A sequence of machines of type $M_1$ can be simulated by a machine of type $M_2$ with advice if there is an encoding mechanism that satisfies Property 4.1. Unfortunately, this is a rather abstract characterization. It was shown that sequences of finite automata with an encoding mechanism satisfying Property 4.30 also satisfy Property 4.1. Similarly, pushdown automata were shown to have an encoding mechanism satisfying Property 4.45, which in turn implied that Property 4.1 was satisfied. For sequences of Turing machines, we used the two-part encoding from the Proof of Proposition 4.21. This encoding consisted of listing $w$, describing a machine to generate tuples and describing the simulating machine $M_{n,w}$. If the machines
are described by transition-lists, then the total description is also a transition-list. This encoding mechanism then satisfies a property similar to Property 4.30 and 4.45. Furthermore, this encoding mechanism satisfies Property 4.1. This indicates that there is a more practical Property that we can use to the same effect.

Property 4.52. Let $M_1$ be a transition-based machine model. Given a sequence of machines of type $M_1$, there is an encoding mechanism for the machines in the sequence and machines $M_{\text{init}}$ and $M_{\text{step}}$ of type $M_2$ such that for an automaton $M$ in the sequence, the description of $M$ using this mechanism has the following properties:

- The description is a transition-list,
- a pointer to the initial state of $M$ can be found by $M_{\text{init}}$ if its data contains the description of $M$,
- for every state $q$ and every configuration $C$ of $M$, which cause the transition function to output state $q'$ and configuration $C'$, if the data of the machine $M_{\text{step}}$ contains the description of $M$ and a pointer to $q$ on this description, as well as a description of $C$, then $M_{\text{step}}$ replaces the configuration $C$ by $C'$ in its data and produces a pointer to $q'$.

Proposition 4.53. Let $M_1$ and $M_2$ be two machine models. Suppose $M_2$ satisfies Property 4.6. Then Property 4.52 implies Property 4.1.

Proof. Let $M_1, M_2, \ldots$ be a sequence of machines of type $M_1$, satisfying Property 4.52. Let $x$ be a string of length $n$ and let $w_n$ be the description of $M_n$. We construct a machine $M$ of type $M_2$ that simulates $M_n$ on $x$. Machine $M$ uses advice $w_n$, so $w_n$ is stored as data. By Property 4.6, $M$ can call the machine $M_{\text{init}}$ to find a pointer to the initial state. Observe that the data of $M$ contains the initial configuration of $M_n$.

Now, $M$ operates in phases. Each phase simulates one transition of $M_n$. A phase starts when $M$ has a pointer to the current state, and its data contains the configuration of $M_n$ up to this point. The machine $M_{\text{step}}$ is called by $M$. Since the data of $M$ contains a description of $M_n$, a pointer to the current state and the configuration of $M_n$, the configuration gets updated correctly and a pointer to the next state is found. When this is done, $M$ is ready to start the next phase.

It follows that $M$ simulates $M_n$ with advice $w_n$. Let $T'$ be the running time of $M_n$, let $S'$ be the space usage of $M_n$ and let $m'$ be the size of $M_n$. Then, the running time $T$ and the space usage $S$ of $M$ depend on the length of the stored data of $M$ $(n, |w_n| \text{ and } S')$, the actual implementations of $M_{\text{init}}$ and $M_{\text{step}}$ (a constant factor) and the number of configurations of $M_n$, which in turn depends on $T'$, $S'$ and $m'$. Thus, for a machine described by $w$ with an input $x$, the functions $T$ and $S$ depend on $|x|, |w|, S', T'$ and $m'$.

We give a similar property for flow-based machine models.

Property 4.54. Let $M_1$ be a flow-based machine model. Given a sequence of machines of type $M_1$, there is an encoding mechanism for the machines in the sequence and machines $M_{\text{step}}$ and $M_{\text{flow}}$ of type $M_2$ such that for an automaton $M$ in the sequence, the description of $M$ using this mechanism has the following properties:
The description is a flow-list,

- a pointer to the next gate of $M$ can be found by $M_{\text{step}}$ if it has the description of $M$ stored as data, and a pointer to the current gate,
- for each gate $q$ of $M$ with edges coming in from gates $q_1, \ldots, q_k$, the output of $q$ can be produced by $M_{\text{flow}}$, if it has the description of $M$, as well as the outputs of gates $q_1, \ldots, q_k$ stored as data, and a pointer to $q$.

We assume that $M_{\text{step}}$ finds the first gate when it has a pointer to the beginning of the description.

**Proposition 4.55.** Let $M_1$ and $M_2$ be two machine models. Suppose $M_2$ satisfies Property 4.4 and 4.6. Then Property 4.54 implies Property 4.1.

**Proof.** Let $M_1, M_2, \ldots$ be a sequence of machines of type $M_1$, satisfying Property 4.54. Let $x$ be string of length $n$ and let $w_n$ be the description of $M_n$. We construct a machine $M$ of type $M_2$ that simulates $M_n$ on $x$. Machine $M$ uses advice $w_n$, so $w_n$ is stored as data. By Property 4.6, $M$ can call the machine $M_{\text{step}}$ to find a pointer to the first gate. From here, $M$ alternately calls the machines $M_{\text{flow}}$ and $M_{\text{step}}$ to find and store all the outputs of the gates, until a stopping criterion is met. It follows that $M$ simulates $M_n$ with advice $w_n$. Observe that $M$ needs to have enough space to store the outputs of all gates of $M_n$, which is possible by Property 4.4.

Let $T'$ be the running time of $M_n$, let $S'$ be the length of the outputs of the gates of $M_n$ and let $m'$ be the size of $M_n$. Then, the running time $T$ and the space usage $S$ of $M$ depend on length of the stored data of $M$ ($n$ and $w_n$), the actual implementations of $M_{\text{step}}$ and $M_{\text{flow}}$ (constant factor), the outputs of the gates ($S'$) and the number of flow steps ($T'$). Thus, the functions $T$ and $S$ depend on $|x|, |w|, S', T'$ and $m'$.

It is not necessarily true that Property 4.1 implies Property 4.52 or 4.54. This is because the machine used in Property 4.1 may use any advice. Thus, advice which behave like characteristic functions can be used, but they convey no information about the internal workings of the machine.

**4.7 Conclusions**

The equivalence result expressed in Theorems 4.12 and 4.13 is stated in such a way that it can be applied to many machine models once the necessary translations have been made. Indeed, the structure of this Chapter is such that this has been done. With these common results in place, the bulk of the work is constructing machines of one class that simulate machines of another. The real strength of the results however, lies in the comparisons that can be made with it. By casting the results in the framework of the abstract properties, it becomes apparent that most results are in fact based on a few very similar core ideas. It enables us to relate the power of sequences of machines to classical advice classes, in several important cases of evolving system models.
Historically, resource bounds of sequences of machines were measured by the sizes of the machines. As we observed, if we relate this measure to advice lengths, then logarithmic blowups occur. This is because the size of a machine is an incomplete measure, its program is also needed. On the other hand, if we use the description size as a resource bound, then no blowups occur (see e.g. Corollaries 4.23 and 4.24 and Theorems 4.41 and 4.51). The description size is a more natural measure for the complexity of a machine (see also Li and Vitányi[31]). Thus, we propose to use description sizes when we discuss sequences of machines. Fortunately, when dealing with logarithmic or polynomial sizes, the blowups have no impact on the resulting sizes.