In this Chapter, we define lineages of automata, a model designed to capture the evolving aspect of computational systems in a natural way. This model, inspired by notions from evolutionary biology, was initially outlined by Van Leeuwen and Wiedermann [30]. It is based on the idea that systems evolve in stages during their existence, with minimal assumptions about the underlying mechanisms. It turns out that even this simple model is more powerful than classical Turing machines, when cast in computational terms. This was initially observed by Van Leeuwen and Wiedermann [28, 30], when they showed that lineages of automata are equivalent to interactive Turing machines with advice (see also Chapter 7). The latter machines are known to possess super-Turing computing power. Here, we develop the theory of lineages in detail.

A lineage is a sequence of interactive, finite automata with a mechanism of passing information from each automaton to its immediate successor and the potential to process infinite input streams. Every automaton in the sequence can be seen as a temporary instantiation of the modeled system, before it changes into the next automaton. We study the properties of lineages through the translations they realize. Lineages of interactive finite automata (or transducers) have been introduced by Van Leeuwen and Wiedermann [30].

The concept of transducers acting on infinite input streams ($\omega$-transducers) is not new. For example, Thomas [41] gives an overview of the theory of finite devices that operate on infinite objects. In the field of non-uniform complexity theory, sequences of computing devices are common-place (see e.g. Balcázar et al. [1] and Chapters 3 and 4). It is the idea of combining these concepts and allowing some form of communication between the devices in the sequence that is new. It allows for a closer modeling of the evolution of a system. The approach leads to several new fundamental questions that will be settled in this Chapter and the next.

The structure of the Chapter is as follows. First, we define lineages. Next, we give some effective constructions to produce new lineages out of given ones.
Then, we look at the class of translations that can be computed by lineages, and show that this class is much richer than any class that is realized by non-evolving finite-state machines. We give a useful characterization of non-uniformly realizable translations in terms of their domain. Finally, we show that the class of translations is closed under composition and under inversion.

Van Leeuwen and Wiedermann\cite{28,30,49,50} showed that several other well-motivated models are equivalent to interactive Turing machines with advice and thus also equivalent to lineages of automata. This implies that lineages are firmly embedded in the family of new models proposed to fill the gap between the classical Turing machine model and its equivalents on the one hand, and the super-Turing behavior of many real-life applications on the other (see Etesi and Németi\cite{12}, Wiedermann and Van Leeuwen\cite{51} and Wegner and Goldin\cite{48} for examples of real-life applications). Between all these new models, lineages stand out because they demonstrate the aspect of evolution directly, rather than indirectly through an advice mechanism. Furthermore, just as (interactive) finite automata are a fundamental model of computation, so are sequences of automata, and hence lineages, a fundamental model of evolving interactive computing.

6.1 Lineages

The building blocks of the model are automata that are a generalization of Mealy automata. These automata process potentially infinite input streams and produce potentially infinite output streams, one symbol at a time. We assume that there is no input tape. Instead, the automaton reads its input from a single input port. One symbol is read from this port at each step. Similarly, the output goes to a single output port, one symbol at a time. In contrast to classical models, the input stream does not have to be known in advance, and can be adjusted at any time by an external agent, based on previous in- and output symbols. This allows the environment to interact with the automaton.

We model the evolutionary aspects by considering sequences of automata. Each automaton in the sequence represents the next evolutionary phase of the system. The way in which this sequence develops need not be described recursively in general. When a transition occurs from one automaton to its successor, the information that the automaton has accumulated over time must be preserved in some way. This is done by requiring that every automaton has a subset of its states in common with its immediate successor.

**Definition 6.1.** An automaton is a 6-tuple $A = (\Sigma, \Omega, Q, I, O, \delta)$, where $\Sigma$ and $\Omega$ are non-empty finite alphabets, $Q$ is a set of states, $I$ and $O$ are subsets of $Q$, and $\delta : Q \times \Sigma \rightarrow Q \times \Omega$ is a (partial) transition function. $\Sigma$ is the input alphabet and $\Omega$ is the output alphabet. We call $I$ the set of entry states and $O$ the set of exit states.

**Definition 6.2.** Let $A$ be a sequence of automata $A_1, A_2, A_3, \ldots$, with $A_i = (\Sigma, \Omega, Q_i, I_i, O_i, \delta_i)$, such that $O_i \subseteq I_{i+1}$ for every $i$. We call $A$ a lineage of automata, or a lineage for short.
The elements in $Q_i - O_i$ are called local states (of $A_i$). The first automaton, $A_1$, has an initial state $q_{in} \in I_1$. Usually, $I_1$ contains only the initial state of $A_1$, and $I_{i+1}$ equals $O_i$. See Figure 6.1 for an example.

6.1 Lineages

**Figure 6.1.** Part of a lineage $\mathcal{A}$. The set of exit states of $A_1$ is a subset of the set of entry states of $A_2$, the set of exit states of $A_2$ is a subset of the set of entry states of $A_3$.

Let $\mathcal{A}$ be a lineage. We assume that $\mathcal{A}$ has one input port and one output port, which is shared between all the automata in the sequence. Assume that we can divide time into discrete time-frames, such that in each time-frame only one symbol appears at the input port and one symbol is produced at the output port. In such a time-frame, exactly one automaton is responsible for processing the input symbol. This automaton is called active at this time. Initially, the first automaton in the sequence is active, and it starts processing the first symbol to appear at the input port. Whenever the currently active automaton $A_i$ enters an exit state $q$, it turns the control over to $A_{i+1}$, which then becomes active. This is done by letting $A_{i+1}$ start processing the next symbols appearing at the input port, beginning in state $q$ (which is an entry state of $A_{i+1}$ by definition). This is called updating, and $A_i$ is the $i$-th update of $\mathcal{A}$.

Again, let $\mathcal{A}$ be a lineage. The entire sequence of symbols that appears at the input port is an input for $\mathcal{A}$. Similarly, the sequence of symbols that is generated at the output port form the output of $\mathcal{A}$. Note that an input is a string in $\Sigma^\infty$ and an output is a string in $\Omega^\infty$. Inputs and outputs are related as follows: let $Q$ be the union of all $Q_i$ and let $x \in \Sigma^\infty$ be an input to a lineage $\mathcal{A}$. Using simultaneous recursion, we define a sequence of states $(q_j)_{j \geq 1}$ in $Q$ and a sequence of integers $(m_j)_{j \geq 1}$, with $m_j$ representing the index of the active automaton at time $j$, as follows:

\[
\begin{align*}
q_1 &= q_{in} , \\
m_1 &= 1 , \\
q_{j+1} &= \pi_1(\delta_{m_j}(q_j, x_j)) , \\
m_{j+1} &= \begin{cases} 
  m_j + 1 & \text{if } q_{j+1} \in O_{m_j} \\
  m_j & \text{otherwise}
\end{cases}.
\end{align*}
\]

(6.1)

Note that $q_{j+1}$ and $m_{j+1}$ depend on $x_{[1:j]}$. Therefore, we also write $q_{j+1}(x_{[1:j]})$ and $m_{j+1}(x_{[1:j]})$ to emphasize the dependence. If $q_j$ is defined for every $j \leq 1 + |x|$, 


then we say that \( x \) is a valid input to \( \mathcal{A} \). In this case, the output of \( \mathcal{A} \) on \( x \) is the string \( y \in \Omega^\infty \) such that \( y_j = \pi_2(\delta_{m_j}(q_j, x_j)) \), for every \( j \geq 1 \).

Let \( \mathcal{A} \) be a lineage of automata and let \( n \) and \( m \) be integers. We say that \( \mathcal{A} \) needs less than \( m \) updates to process all strings of length \( n \) if \( m_n(x) \leq m \) for every string \( x \) of length \( n \).

**Definition 6.3.** Let \( \mathcal{A} \) be a lineage. We define the partial function \( \Phi^\mathcal{A} : \Sigma^\infty \rightarrow \Omega^\infty \) by letting \( \Phi^\mathcal{A}(x) \) be the output of \( \mathcal{A} \) on \( x \) if \( x \) is a valid input and undefined otherwise, for every string \( x \). We say that \( \Phi^\mathcal{A} \) is non-uniformly realized by the lineage \( \mathcal{A} \). In general, for a partial function \( \Psi : \Sigma^\infty \rightarrow \Omega^\infty \), we say that \( \Psi \) is non-uniformly realizable, if there is a lineage \( \mathcal{A} \) such that \( \Psi \) equals \( \Phi^\mathcal{A} \).

For many lineages \( \mathcal{A} \), the translation \( \Phi^\mathcal{A} \) is not realizable by a single finite-state transducer and not even for a Turing transducer (Proposition 6.14). See also Van Leeuwen and Wiedermann[30].

### 6.2 Constructions on Lineages

In this section, we give some methods to construct new lineages \( \mathcal{B} \) out of a given lineage \( \mathcal{A} \) that non-uniformly realize the same translation, i.e., such that \( \Phi^\mathcal{B} \) equals \( \Phi^\mathcal{A} \). In fact, we show two extreme cases: a method that postpones updates of automata to arbitrary finite times and a method that updates as often as possible, i.e., after each step.

To distinguish between states of different automata (in a lineage), we let \( Q^\mathcal{A} \) be the set of states, \( I^\mathcal{A} \) the set of entry states and \( O^\mathcal{A} \) the set of exit states of an automaton \( \mathcal{A} \).

#### 6.2.1 Merging Two Successive Automata in a Lineage

The first method merges two successive automata \( A_i \) and \( A_{i+1} \) into one new automaton \( B_i \) such that \( A_i \) followed by \( A_{i+1} \) translates input segments in the same way as \( B_i \). To obtain a new lineage \( \mathcal{B} \) that non-uniformly realizes the same translation as \( \mathcal{A} \), we let \( B_j = A_j \) for all \( j < i \), and we let \( B_j = A_{j+1} \) for all \( j > i \).

**Construction 6.1.** Let \( Q^{B_i} \) be the disjoint union of \( Q^{A_i} \) and \( Q^{A_{i+1}} \), that is,

\[
Q^{B_i} = \{ (q, i) \mid q \in Q^{A_i} \} \cup \{ (q, i + 1) \mid q \in Q^{A_{i+1}} \}.
\]  

(6.2)

Roughly speaking, a state \((q, i)\) corresponds to a state \( q \) in \( A_i \), while a state \((q, i+1)\) corresponds to a state \( q \) in \( A_{i+1} \). Note that each exit state \( q \) of \( A_i \) has two copies in \( Q^{B_i} \), namely \((q, i)\) and \((q, i+1)\). If \( q \) is not an exit state of \( A_{i+1} \), then both copies can be local states, but if \( q \) is an exit state of \( A_{i+1} \), then one (and only one) of the copies is an exit state. In this case we let \((q, i)\) be the exit state. Thus we define

\[
I^{B_i} = \{ (q, i) \mid q \in I^{A_i} \} \cup \{ (q, i + 1) \mid q \in O^{A_{i+1}} - O^{A_i} \} \cup \{ (q, i) \mid q \in O^{A_{i+1}} \cap O^{A_i} \}.
\]  

(6.3)

Note that each exit state \( q \) of \( A_{i+1} \) has only one copy in \( Q^{B_i} \), namely \((q, i+1)\) if it is an exit state, and \((q, i)\) if it is not. Each entry state \( q \) of \( A_i \) has only one copy in \( Q^{B_i} \), namely \((q, i)\) if it is an entry state, and \((q, i + 1)\) if it is not.
Let $\delta_i$ and $\delta_{i+1}$ be the transition functions of $A_i$ and $A_{i+1}$, respectively. We define the transition function $\gamma_i$ of $B_i$ as follows:

For $q \in Q^{A_i}$ and $a \in \Sigma$, the transition $\gamma_i((q, i), a)$ is defined by the following cases:

- If $\delta_i(q, a) = (r, b)$ and $r \notin O^{A_i}$, then $\gamma_i((q, i), a) = ((r, i), b)$,
- if $\delta_i(q, a) = (r, b)$ and $r \in O^{A_i}$, then $\gamma_i((q, i), a) = ((r, i+1), b)$ and
- if $\delta_i(q, a)$ is undefined, then so is $\gamma_i((q, i), a)$.

For $q \in Q^{A_{i+1}}$ and $a \in \Sigma$, the transition $\gamma_i((q, i+1), a)$ is defined by these cases:

- If $\delta_{i+1}(q, a) = (r, b)$ and $r \notin O^{A_i} \cap O^{A_{i+1}}$, then $\gamma_i((q, i+1), a) = ((r, i+1), b)$,
- if $\delta_{i+1}(q, a) = (r, b)$ and $r \in O^{A_i} \cap O^{A_{i+1}}$, then $\gamma_i((q, i+1), a) = ((r, i), b)$,
- if $\delta_{i+1}(q, a)$ is undefined, then so is $\gamma_i((q, i+1), a)$.

Note that an exit state $(r, i)$ with $r \in O^{A_i} \cap O^{A_{i+1}}$ cannot be entered from a state $(q, i)$. To make sure that the exit states of $B_i$ are entry states of $B_{i+1}$, we should relabel every state $q$ of $B_i$ as $(q, i+1)$ (unless $q$ is an exit state of both $A_i$ and $A_{i+1}$, in which case $(q, i)$ is the correct label). The transition function has to be adjusted accordingly. Similarly, every state $q$ of $B_{i-1}$ should be relabeled $(q, i)$. A similar relabeling has to occur for every automaton $B_j$, with $j \neq i$. See Figure 6.2 for an example.

![Figure 6.2](image)

**Figure 6.2.** In the lineage $A$ from Figure 6.1, the automata $A_1$ and $A_2$ are replaced by the automaton $B_1$.

**Proposition 6.4.** Lineage $B$ from Construction 6.1 non-uniformly realizes the same translation as $A$.

**Proof.** Since the states of $B_j$ have been relabeled for $j < i$, the automaton $B_i$ starts in an entry state $(q, i)$ iff $A_i$ starts in $q$.

To see that $B_i$ translates input segments in the same way as $A_i$ and $A_{i+1}$ combined, consider a string $x$. Suppose $A_i$ starts in a state $q_1$ and processes a part of $x$, until it enters an exit state $q_3$, say after $n_i$ symbols of $x$. At this point, $A_{i+1}$ processes the remainder of $x$, starting in $q_3$. Let $q_2$ be the state that $A_i$ was in before entering $q_3$. Now observe the action of $B_i$ on $x$. It starts in $(q_1, i)$, and processes $x$ in exactly the same way as $A_i$ for $n_i - 1$ steps, ending up in $(q_2, i)$. The next transition goes to $(q_3, i+1)$.
Suppose $A_{i+1}$ enters an exit state $q_5$ after processing another part of $x$, say of length $n_{i+1}$. Let $q_4$ be the state $A_{i+1}$ was in just before entering $q_5$. From $(q_3, i+1)$, the automaton $B_i$ ends up in $(q_4, i + 1)$ after $n_{i+1} - 1$ steps. If $q_5$ is also an exit state of $A_i$, then the next transition goes to $(q_5, i)$, which is an exit state of $B_i$ by definition. Otherwise, the next transition goes to $(q_5, i + 1)$, which is also an exit state. Either way, $B_i$ enters the exit state corresponding to $q_5$. Since the states of $B_j$ have been relabeled for $j > i$, the rest of $x$ is processed correctly.

If either $A_i$ or $A_{i+1}$ does not enter an exit state, then the transitions that occur will be mimicked by $B_i$ (with either $i$ or $i + 1$ resp. appended to the state). It follows that $B$ non-uniformly realizes the same translation as $A$.

Construction 6.1 can be applied repeatedly to merge any fixed number of consecutive automata. The intuitive notion behind lineages is that they model the evolution of a system. Applying Construction 6.1 to a lineage can be thought of as “making bigger jumps in the evolution of the system”.

### 6.2.2 Updating the Lineage at Each Step

The next method turns a lineage $A$ into a lineage $B$ that non-uniformly realizes the same translation, in such a way that each automaton only processes one input symbol, i.e., after every single step the active automaton is updated to the next one.

**Construction 6.2.** First, we let the set of states for the lineage $B$ be

$$\{ (q, i) \mid q \in A_i \}.$$  \hspace{1cm} (6.4)

Now, we recursively construct the automaton $B_n$. Let $q_{i_n}$ be the initial state of $A_1$. Then the initial state of $B_1$ is the state $(q_{i_1}, 1)$, and $I^{B_1} = \{(q_{i_1}, 1)\}$. Suppose the set of entry states $I^{B_n}$ of $B_n$ has been constructed. Then the set of output states $O^{B_n}$ consists of all the states that are reachable from a state in $I^{B_n}$ in one step, and we let $Q^{B_n} = I^{B_n} \cup O^{B_n}$. Let’s make this more formal. Let $(q, i)$ be a state in $I^{B_n}$, let $a \in \Sigma$, and let $\delta_i$ be the transition function of $A_i$. Suppose $\delta_i(q, a)$ is defined. Then there is a $b \in \Sigma$ and a state $r$ such that $\delta_i(q, a) = (r, b)$. If $r$ is an exit state of $A_i$, then we let $i' = i + 1$, otherwise $i' = i$. We add the state $(r, i')$ to $O^{B_n}$, and define $\gamma_n \left((q, i), a\right) = ((r, i'), b)$, where $\gamma_n$ is the transition function of $B_n$. We do this for each state $(q, i)$ in $I^{B_n}$ and every $a \in \Sigma$. Once $O^{B_n}$ is constructed, we construct the set of entry states of $B_{n+1}$ by defining $I^{B_{n+1}} = O^{B_n}$.

In each automaton, all transitions go from an entry state to an exit state. This means that, after reading one input symbol, the automaton is updated.

**Proposition 6.5.** Lineage $B$ from Construction 6.2 non-uniformly realizes the same translation as $A$.

**Proof.** Let $x$ be an input string. It is left to the reader to prove, using induction, that prior to reading the $n$-th input symbol, $A_i$ is active in state $q$ iff $B_n$ is active in state $(q, i)$. By inspecting the transition functions, we see that both automata will output the same symbol when they process $x_n$. So $B$ non-uniformly realizes the same translation as $A$. \hfill $\square$
Applying Construction 6.2 to a lineage can be thought of as “taking smaller steps in the evolution of the system”.

6.2.3 Reducing the Number of States

The next method works only on lineages that update after every input symbol. From section 6.2.2, we conclude that every lineage is equivalent to one of this type and that in such a lineage, a state is either an entry state, or an exit state, or both. This means that the total number of states of the \(n\)-th update is at least \(\max\{|I^A_n|, |O^A_n|\}\). We will alter the lineage such that the total number of states of \(B_n\) equals this maximum.

**Construction 6.3.** Let \(Q\) be an infinite set of states. Recursively construct injective functions \(f_n : I^A_n \to Q\) and \(g_n : O^A_n \to Q\) such that \(f_{n+1}(q) = g_n(q)\) for every state in \(O^A_n\), and \(|f_n(I^A_n) \cup g_n(O^A_n)| = \max\{|I^A_n|, |O^A_n|\}\). The actual construction of \(f_n\) and \(g_n\) is left to the reader. Construct a lineage \(B\) such that the set of entry states of \(B_n\) is the set \(f_n(I^A_n)\), and the set of exit states is \(g_n(O^A_n)\). Let \(q_{\text{in}} \in I^A_1\) be the initial state of \(A_1\). Then \(f_1(q_{\text{in}})\) is the initial state of \(B_1\).

Let \(q\) be a state in \(I^A_n\) and \(a \in \Sigma\), and let \(\delta_n\) be the transition function of \(A_n\). If \(\delta_n(q, a)\) is defined, then there is an exit state \(r\) and a \(b \in \Sigma\) such that \(\delta_n(q, a) = (r, b)\). Now define \(\gamma_n(f_n(q), a) = (g_n(r), b)\), where \(\gamma_n\) is the transition function of \(B_n\). Since \(f_n\) and \(g_n\) are injective, this definition is unambiguous.

**Proposition 6.6.** Lineage \(B\) from Construction 6.3 non-uniformly realizes the same translation as \(A\).

**Proof.** Let \(x\) be an input string. It is left to the reader to prove that, prior to reading the \(n\)-th input symbol, \(A_n\) is in state \(q\) iff \(B_n\) is in state \(f_n(q)\). Just as in the previous method, we can conclude from this fact that \(B\) non-uniformly realizes the same translation as \(A\).

\(\square\)

We summarize the last two constructions in the following result:

**Proposition 6.7.** If a translation is non-uniformly realizable, then it can be non-uniformly realized by a lineage \(B\) with the property that its automata \(B_i\) update after every step and have precisely \(\max\{|I^{B_i}|, |O^{B_i}|\}\) states each.

6.3 Properties of Non-uniformly Realizable Translations

In this section, we show the basic properties of the class of translations that are non-uniformly realized by lineages. First, we give a useful characterization of non-uniformly realizable translations in terms of their domains. Next, we show that the class is uncountable and contains non-recursive translations. Then, we show that the class is closed under composition and inversion.
6.3.1 A Characterization of Non-uniformly Realizable Translations

It is possible to characterize the translations that are non-uniformly realized by a lineage without actually constructing lineages, by specializing the theory of continuous mappings (see Staiger[39]). This very useful characterization depends on the domain on which the translation is defined and the relation it specifies between input and output pairs.

**Theorem 6.8.** A translation $\Phi$ with domain $D$ can be non-uniformly realized by a lineage $A$ iff

- $|\Phi(x)| = |x|$ for all $x \in D$,
- if $u$ is a prefix of $x \in D$, then $u \in D$ and $\Phi(u)$ is a prefix of $\Phi(x)$,
- $D$ is closed.

We prove the Theorem with the following Propositions.

**Proposition 6.9.** Let $\Phi$ be a non-uniformly realizable translation with domain $D$. Then $|\Phi(x)| = |x|$ for all $x \in D$.

_Proof._ This follows directly from (6.1) and Definition 6.3.

**Proposition 6.10.** Let $\Phi$ be a non-uniformly realizable translation with domain $D$. If $u$ is a prefix of $x \in D$, then $u \in D$ and $\Phi(u)$ is a prefix of $\Phi(x)$.

_Proof._ Let $\Phi$ be non-uniformly realized by the lineage $A$. Let $x \in D$. Then $x$ is a valid input to $A$. If $u$ is a prefix of $x$, then $u$ is also a valid input. The result then follows from (6.1) and Definition 6.3.

**Proposition 6.11.** Let $\Phi$ be a non-uniformly realizable translation with domain $D$. Then $D$ is closed.

_Proof._ Let $A$ be a lineage that non-uniformly realizes $\Phi$. Let $x \notin D$ be a string and consider a run of $A$ on $x$. Because $x$ is not in the domain of $\Phi$, there is a finite prefix $u$ of $x$, that is not processed by $A$.

Let $y$ be a string in $B(u)$ and consider a run of $A$ on $y$. We conclude that $y$ cannot be processed by $A$. It follows that $B(u)$ does not intersect $D$, which implies that $D$ is a closed set.

The previous Propositions showed that a non-uniformly realizable translation fulfills the conditions of Theorem 6.8. The next Proposition constructs a lineage for a translation that satisfies the conditions. For a domain $D$, denote the set $D \cap \Sigma^n$ by $D_n$.

**Proposition 6.12.** Let $\Phi$ be a translation with domain $D$. Suppose that

- $|\Phi(x)| = |x|$ for all $x \in D$,
- if $u$ is a prefix of $x \in D$, then $u \in D$ and $\Phi(u)$ is a prefix of $\Phi(x)$,
- $D$ is closed.
Then $\Phi$ can be non-uniformly realized by a lineage $A$ that updates after every step, such that $A_n$ has $|D_{n-1}|$ entry states and $|D_n|$ exit states.

**Proof.** Define the set of states of $A_n$ for $n \geq 1$ by

$$
I_n = \{ [u] \mid u \in D_{n-1} \} ,
$$

$$
O_n = \{ [u] \mid u \in D_n \} ,
$$

and let $Q_n = I_n \cup O_n$. The initial state of $A_1$ is $[\varepsilon]$. The transition function $\delta_n$ is defined by

$$
\delta_n([u], a) = \begin{cases} 
([ua], (\Phi(ua))_n) & \text{if } ua \in D_n \\
\text{undefined} & \text{otherwise}
\end{cases}.
$$

The transition function is well-defined, since $D_n$ is a subset of $D$ and $|\Phi(x)| = |x|$ for all $x$. Using induction and the fact that $\Phi(u)$ is a prefix of $\Phi(ua)$ for $u, ua \in D$, one can show that $A$ produces $\Phi(x)$ on input $x \in D$.

Suppose on the other hand that $x \not\in D$. Since $D$ is closed, there is a basis set $B(u)$ that contains $x$, which does not intersect $D$. It follows that $u \not\in D$. Since the transition functions are not defined on $u$, we see that $u$ (and therefore $x$) is not a valid input to $A$. Hence $A$ non-uniformly realizes $\Phi$.

For an example of a translation that cannot be non-uniformly realized by a lineage, consider the translation from Example 6.13.

**Example 6.13.** Let $\Sigma = \{0, 1\}$, and $\Omega = \{a, b, c, d\}$. Define the help-function $\psi$ by:

$$
\psi(00) = aa ,
\psi(01) = bb ,
\psi(10) = cc ,
\psi(11) = dd .
$$

(6.7)

Now, we define the translation $\Phi : \Sigma^\infty \rightarrow \Omega^\infty$ by

$$
\Phi(x) = \psi(x_1x_2)\psi(x_3x_4)\ldots ,
$$

(6.8)

for strings $x$ of infinite length. If $x$ is a finite string, then $\Phi(x)$ is undefined.

Since $\Phi$ is not defined on finite prefixes, it is not non-uniformly realizable. Furthermore, it can not be embedded into a non-uniformly realizable translation. To see this, suppose that there is a lineage $A$ which behaves like $\Phi$ on infinite inputs. Consider the possibilities when $A_1$ is given the input 0. It must produce an output. If $a$ is produced, then $A$ fails to correctly process strings that start with 01, but if it doesn’t produce $a$, strings which start with 00 are not properly processed. We conclude that $\Phi$ cannot be embedded in a non-uniformly realizable translation.

### 6.3.2 The Number of Non-uniformly Realizable Translations

Observe that any translation that is realized by a finite-state transducer can also be non-uniformly realized by a lineage (just take infinitely many copies of the transducer that realizes the translation). On the other hand, the class of non-uniformly
realizable translations contains uncountably many translations that cannot be realized by a finite-state transducer. We deduce that the class of non-uniformly realizable translations is uncountable, whereas the class of translations that are realized by finite-state transducers is countable.

**Proposition 6.14.** Let Ω be an alphabet with at least two elements. There are uncountably many non-uniformly realizable translations from Σ (any Σ) to Ω that cannot be realized by a finite-state transducer.

*Proof.* Pick two elements of Ω, call them 0 and 1. Let N be a set of positive integers that is not recursively enumerable and let \((n_i)_{i \geq 1}\) be an enumeration of N. Consider the infinite string \(y = 1^{n_1}01^{n_2}01^{n_3}0\ldots\). Define the translation \(\Phi\) by letting \(\Phi(x)\) be a prefix of \(y\) of length \(|x|\), for every \(x\) in \(\Sigma^\infty\). Since the domain is all of \(\Sigma^\infty\), it fulfills the conditions of Proposition 6.12, so \(\Phi\) is non-uniformly realizable.

Suppose that \(\Phi\) can be realized by an automaton \(A\). Let \(a\) be a letter in \(\Sigma\) and let \(M\) be a Turing machine, on input \(i \in \mathbb{N}\) in unary, simulates \(A\) on input \(a^*\) until \(A\) has written \(i\) zeroes. Then \(M\) outputs the last sequence of ones in \(A\)'s output. We see that \(M\) computes \(n_i\) in unary. It follows that \(M\) enumerates \(N\), which is a contradiction. Because there are uncountably many sets that are not recursively enumerable, we have the desired result. \(\square\)

If in the proof we replace the automaton \(A\) by a Turing machine, the proof is still valid. We conclude that lineages possess super-Turing computing power.

**6.3.3 Composition of Non-uniformly Realizable Translations**

The general class of translations is closed under the operations of composition and inverse. A natural question that arises, is whether the class of non-uniformly realizable translations is also closed under these operations. This question is answered in this and the next subsection.

**Proposition 6.15.** Let \(\Phi^A : \Sigma^\infty \to \Omega^\infty\) and \(\Phi^B : \Omega^\infty \to \Theta^\infty\) be translations non-uniformly realized by lineages \(A\) and \(B\) respectively. Then a lineage \(C\) exists such that \(\Phi^C = \Phi^B \circ \Phi^A\).

*Proof.* Given lineages \(A\) and \(B\), we construct a lineage \(C\) by defining for every automaton \(C_i\) its set of states, its initial state and its transition function as follows. The set of states of \(C_i\) is defined by:

\[
Q^{C_i} = \{ (q, k, r, l) \mid k, l \leq i + 1, \ q \in Q^A_k \land r \in Q^B_l \}. \tag{6.9}
\]

When \(C_i\) is in state \((q, k, r, l)\), this simulates the fact that \(A_k\) has entered state \(q\) and \(B_l\) has entered state \(r\). So, \(C\) simulates the transitions of \(A\) in the first two components of its states and the transitions of \(B\), with the output of \(A\) as input, in the last two components. Special care must be taken with exit states. If \(A_k\) enters the exit state \(q\), then \(A_{k+1}\) will start in state \(q\). So, the corresponding state will be \((q, k + 1, \ldots, \ldots)\). A similar thing holds for \(B\).
Let $q_{in}$ be the initial state of $A_1$ and $r_{in}$ the initial state of $B_1$. The initial state of $C_1$ is $(q_{in}, 1, r_{in}, 1)$. A state $(q, k, r, l)$ is an exit state if $q$ is an exit state of $A_{k-1}$ (and $k > 1$) or $r$ is an exit state of $B_{l-1}$ (and $l > 1$). The state is an entry state if $q$ is an entry state of $A_k$ or $r$ is an entry state of $B_l$.

Now, we define the transition function $\tau_i$ of $C_i$. Let $(q, k, r, l)$ be a state in $C_i$, with $k, l \leq i$, and let $a$ be a letter. Let $\delta_k$ be the transition function of $A_k$ and $\gamma_l$ the transition function of $B_l$. Suppose that $\delta_k(q, a) = (q', b)$ and $\gamma_l(r, b) = (r', c)$. The transition function $\tau_i$ simulates the changes in $A$ and $B$. In $A$, the parameters $q$ and $k$ change to $q'$ and $k'$, with $k' = k + 1$ if $q'$ is an exit state, or $k' = k$. Similarly, in $B$, the parameters $r$ and $l$ change to $r'$ and $l'$. Thus, we let $\tau_i((q, k, r, l), a) = ((q', k', r', l'), c)$. In all other cases we let $\tau_i$ be undefined.

Proving that $C$ non-uniformly realizes the translation $\Phi^B \circ \Phi^A$ is similar to the proof of Proposition 6.5. If $x$ is an input to $A$ and $y = \Phi^A(x)$ is an input to $B$ then, just before reading the $n$-th input symbol, $C_i$ is in state $(q, k, r, l)$ iff $A_k$ is in state $q$ and $B_l$ is in state $r$. Inspecting the transition functions, we see that in this case $\Phi^B(\Phi^A(x)) = \Phi^C(x)$. If, on the other hand, either $x$ or $y$ is not valid for $A$ or $B$ respectively, then there comes a time when either $\delta_k(q, x_n)$ or $\gamma_l(r, y_n)$ is undefined. In either case, $\tau_i((q, k, r, l), x_n)$ is also undefined. Hence the domains match and the two functions are equal.

\[ \Box \]

The set of states $Q_{C_1}$ in the given proof can be taken much smaller. In fact, we don’t need the states $(q, i + 1, \_ , \_ )$ unless $q$ is an exit state of $A_i$. Likewise we can do without the states $(\_ , \_ , r, i + 1)$ if $r$ is not an exit state of $B_i$.

### 6.3.4 The Inverse of a Non-uniformly Realizable Translation

Much of the theory presented up until now, remains valid when we ignore finite inputs and consider translations as functions from infinite strings to infinite strings only. The results in this section however, can not be carried over. Consider Example 6.16.

**Example 6.16.** Let $\Sigma = \{a, b, c, d\}$, and $\Omega = \{0, 1\}$. Define the help-function $\psi$ by:

\[
\begin{align*}
\psi(aa) &= 00, \\
\psi(bb) &= 01, \\
\psi(cc) &= 10, \\
\psi(dd) &= 11.
\end{align*}
\]  

(6.10)

The output of $\psi$ is undefined for all other inputs. Now, we define the translation $\Phi : \Sigma^\omega \to \Omega^\omega$ by

\[
\Phi(x) = \psi(x_1x_2)\psi(x_3x_4)\psi(x_5x_6)\ldots.
\]  

(6.11)

This translation is injective. Although it does not fit the conditions of Proposition 6.12, it can be embedded into a non-uniformly realizable translation. Thus, when one ignores finite inputs, one might say that $\Phi$ is an injective non-uniformly realizable translation. The inverse of $\Phi$ is the translation from Example 6.13. We see that although $\Phi$ can be embedded into a non-uniformly realizable translation, its inverse cannot.
Note that the non-uniformly realizable translation used for the embedding is not an injection, since $\Phi(a) = \Phi(b) = 0$. The following Proposition shows that for injective non-uniformly realizable translations (where finite inputs are taken into account), their inverses are non-uniformly realizable.

**Proposition 6.17.** Let $\Phi$ be an injective non-uniformly realizable translation with domain $D$. Then the translation $\Phi^{-1}$ with domain $\Phi(D)$ can be non-uniformly realized.

**Proof.** Let $A$ be a lineage that non-uniformly realizes $A$. We will transform $A$ into a lineage $B$ that non-uniformly realizes $\Phi^{-1}$. Consider the automaton $\mathcal{A}_k$, with transition function $\delta_k$. First, we remove all states that cannot be reached from the initial state.

Suppose $\delta_k(q, a) = (r, b)$ and $\delta_k(q, a') = (r', b)$. Let $u$ be a string such that $A_k$ enters $q$ after processing $u$, with output $v$. Then the output belonging to $ua$ is $vb$, and the output belonging to $ua'$ is also $vb$. Since $\Phi$ is injective, it follows that $a = a'$ (and $r = r'$). Therefore the function $\gamma_k$, defined by

$$
\gamma_k(q, b) = \begin{cases} (r, a) & \text{if } \delta_k(q, a) = (r, b) \\ \text{undefined otherwise} \end{cases},
$$

(6.12)
is well-defined. The automaton $B_k$ is defined by taking $A_k$, with $\delta_k$ replaced by $\gamma_k$.

By inspecting the transition functions, we see that for strings $y = \Phi(x)$, the lineage $B$ gives $x$ as output.

Let $y$ be a string not in $\Phi(D)$. Then there is a largest prefix $v$ of $y$, such that $A$ gives $v$ as output, on an input $u$. Suppose $A$ enters a state $q$ after processing $u$. There is no transition from $q$ that gives $y|_{v|+1}$ as output. It follows that there is no transition from $q$ with input $y|_{v|+1}$ in $B$. Then, $B$ enters $q$ after processing $v$, but it cannot process $y|_{v|+1}$, thus $y$ is not a valid input to $B$. Hence the domain of $B$ is $\Phi(D)$. We conclude that $B$ non-uniformly realizes $\Phi^{-1}$.

\[\Box\]

Most translations can be made injective by just restricting their domain. If a translation is a bijection, then it follows that the input and output alphabets need to be of the same size. Propositions 6.15 and 6.17 show that the class of non-uniformly realizable translations is closed under composition and inversion.

### 6.4 Conclusions

In this Chapter, lineages of automata were introduced. This model was introduced in a slightly different form by Van Leeuwen and Wiedermann[30]. This model is different from other sequence-based models such as the ones in Chapter 4 by the fact that the automata in a lineage have a method of passing information to their immediate successor. By passing along enough information, the computation of an automaton can then be continued by its successor. Another way of looking at it is that the automaton changes or evolves into its immediate successor.
Since we are dealing with automata, all the information has to be stored in the finite control. Information is passed along by using sets of states that are shared between two neighboring automata in a sequence. The entry states of an automaton in lineage can be compared to initial states of a classical finite automaton. The exit states basically force an automaton to halt. The next automaton can then resume the computation. Thus, the exit states can be seen as another halting criterion.

We gave several constructions to produce new lineages out of given ones. It is important to note that all these constructions were recursive procedures. The translations that are non-uniformly realized by lineages can be characterized in terms of their domains and the relations between inputs and outputs. This characterization does not use the lineages at all, so one can describe the translations without constructing a lineage. It proves much easier to test if a translation is non-uniformly realizable by using this characterization then it is to define a lineage that non-uniformly realizes the translation.