Alcom-FT Technical Report Series
ALCOMFT-TR-03-195

Lineages of Automata ⋆

A Model for Evolving Interactive Systems.

Peter Verbaan[1] Jan van Leeuwen[1] Jiří Wiedermann[2]

[1] Institute of Information and Computing Sciences, Utrecht University, the Netherlands
[2] Institute of Computer Science, Academy of Sciences of the Czech Republic, Czech Republic

Abstract. While in the series of previous papers we designed and studied a number of models of evolving interactive systems, in the present paper we concentrate on an in-depth study of a single model that turned out to be a distinguished model of evolving interactive computing: lineages of automata. A lineage consists of a sequence of interactive finite automata, with a mechanism of passing information from each automaton to its immediate successor. In this paper, we develop the theory of lineages. We give some means to construct new lineages out of given ones and prove several properties of translations that are realized by lineages. Lineages enable a definition of a suitable complexity measure for evolving systems. We show several complexity results, including a hierarchy result. Lineages are equivalent to interactive Turing machines with advice, but they stand out because they demonstrate the aspect of evolution explicitly.

1 Introduction

In recent years, several people have come to the conclusion that the Turing machine model, which has long been the basis for theoretical computer science, fails to capture all the characteristics of modern day computing (cf. [10]). When we think of a modern networked computing system, we see a device which is capable of interaction with its environment and can be changed over time (by installing new hardware or upgrading the software). A computation on such a device can be extended arbitrarily, which implies the existence of potentially infinite computations. Networked machines and the programs that run on them are examples of interactive evolving systems ([4, 6]). Further examples are given in [11, 12].

If we look at a (deterministic) computation (or a transduction) from a traditional point of view, the entire input, as well as the quantity of available resources, is fixed the moment we start the computation. In a more realistic setting, we want a user to generate the input interactively and allow the computing device to allocate more resources when the need arises. We might even allow the user to alter the device's behaviour during the computation. Last but not least, we should consider potentially never-ending computations. Devices that allow these kinds of computations are called *evolving interactive systems*. In this paper, we define lineages of automata, a simple yet elegant model which captures the evolving aspect of computational systems in a natural way. It turns out that even this simple model is more powerful than classical Turing machines. This has been merely sketched in [4, 6], where lineages of automata have been shown to be equivalent to so-called interactive Turing machines with advice. The latter machines are known to possess super-Turing computing power.

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

⋆ Version December 5, 2003. This research was partially supported by GA ČR grant No. 201/02/1456 and by EC Contract IST-1999-14186 (Project ALCOM-FT).

input streams. Every automaton in the sequence can be seen as a temporary instantiation of the system, before it changes into a different automaton. We study the properties of lineages through the translations they realize. Lineages of interactive finite automata (or: transducers) have been introduced in [6].

The concept of transducers acting on infinite input streams (ω-transducers) is not new. For example, [9] 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 ([1]). It is the idea of combining these concepts and allowing some form of communication between the devices in the sequence that is new. The approach leads to several new fundamental questions that will be settled in this paper.

We can measure the "speed of growth" (i.e. "growth complexity") of a lineage by a function that relates the index of each automaton to its size. That is, the complexity of a lineage is a function g such that the n-th automaton in the sequence has $g(n)$ states. Using this measure, we can divide the translations computed by evolving systems into classes based on the complexity of the lineages that realize them. Our main result states that this division is non-trivial, i.e. for every positive, non-decreasing function g, there is a translation that can be realized by a lineage with complexity g, but not by any lineage with a lower complexity.

The structure of the paper is as follows. In section 2, we define lineages. Next, in section 3, we give some effective constructions to produce new lineages out of given ones. These constructions, and the lemma's that accompany them, illustrate a key idea that is used extensively in this paper: to associate (input-)prefixes with states. In section 4, we look at the class of translations that can be realized 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 characterisation of l-realizable translations in terms of the domain and the relation between input and output pairs. We also show that the class is closed under composition and, with minor restrictions, under inversion. Then, in section 5, we define a novel measure of complexity on translations and establish a hierarchy of complexity classes. Finally, in section 6, we show that lineages are "exponentially" equivalent to interactive Turing machines with advice.

In [4, 6, 11, 12], several other well-motivated models are shown to be equivalent to interactive Turing machines with advice. This implies that lineages are firmly embedded in the family of new models proposed to fill the gap between the original Turing machine model and its equivalents on the one hand, and real-life applications on the other [10]. Between all these new models, lineages stand out because they demonstrate the aspect of evolution directly, rather than indirectly through an advice mechanism. The growth and rate of growth measures are also very easily expressible with lineages. 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.

1.1 Notation

In most literature, the term transducer is used to denote an automaton with output capability. The main reason to introduce this term is to distinguish it from automata with, and automata without an output mechanism. In this paper, we only regard automata with output. Thus, every time we use the word automaton, the term transducer could be substituted without ill effects.

We use Σ and Ω to denote alphabets. We use the notations $\Sigma^\star$, Σ^ω for the sets of finite and infinite strings over Σ respectively and Σ^∞ for the union of $\Sigma^\star$ and Σ^ω. We call a partial function from Σ^∞ to Ω^∞ a *translation*.

For a string $x \in \Sigma^\infty$ of length at least n, we denote the n-th symbol of x by x_n, or $(x)_n$, to improve readability. We write $x_{[i:j]}$ for $x_i x_{i+1} \dots x_j$. We also use the projection functions for tuples, π_n, which are defined straightforwardly, i.e., π_n 'returns' the n-th component of a tuple that serves as the argument of π_n, for any n.

Let D be a subset of Σ^∞. We define the n-th *prefix domain* $P^n(D)$ as the set of all prefixes of length n of strings in D,

$$P^n(D) = \{\ x_{[1:n]} \quad | \quad x \in D\ \}\ . \tag{1}$$

We define a topology on Σ^∞ as follows. Let u be a finite string over Σ. Then the set of all possible extensions of u,

$$\mathbb{B}(u) := \{\ x \in \Sigma^\infty \quad | \quad u \text{ is a prefix of } x\ \} \tag{2}$$

is a *basis set*. Let S be a subset of Σ^∞. We call S an *open set* if it is a union of basis sets. A set is *closed* if it is the complement of an open set.

2 Modelling Evolutionary Interactive Systems by Lineages

As we explained in the introduction, the ideas of classical computability theory are not sufficient to capture all aspects of modern computing systems. This validates the research into theories that better describe these systems. In this paper, we develop a new model for computing systems, initially outlined in [6]: lineages, inspired by a similar notion in evolutionary biology.

The building blocks of the model are a generalisation 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 1. *An automaton is a 6-tuple $A = (\Sigma, \Omega, Q, I, O, \delta)$, where Σ and Ω are non-empty finite alphabets, Q is a set of states, I and O are subsets of Q, and $\delta : Q \times \Sigma \to Q \times \Omega$ is a (partial) transition function. Σ is the input alphabet, and Ω is the output alphabet. We call I the set of* entry states *and O the set of* exit states*.*

Definition 2. *Let $\mathcal{A}$ be a sequence of automata $A_1, A_2, \dots,$ 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 $\mathcal{A}$ a* lineage of automata*, or a lineage for short.*

We do not require a recursive recipe for defining or constructing the sequences. The elements in $Q_i - O_i$ are called *local states* (of A_i). The first automaton, A_1, has an initial state $q_{\text{in}} \in I_1$. Usually, I_1 contains only the initial state of A_1, and I_{i+1} equals O_i. See Fig. 1 for an example.

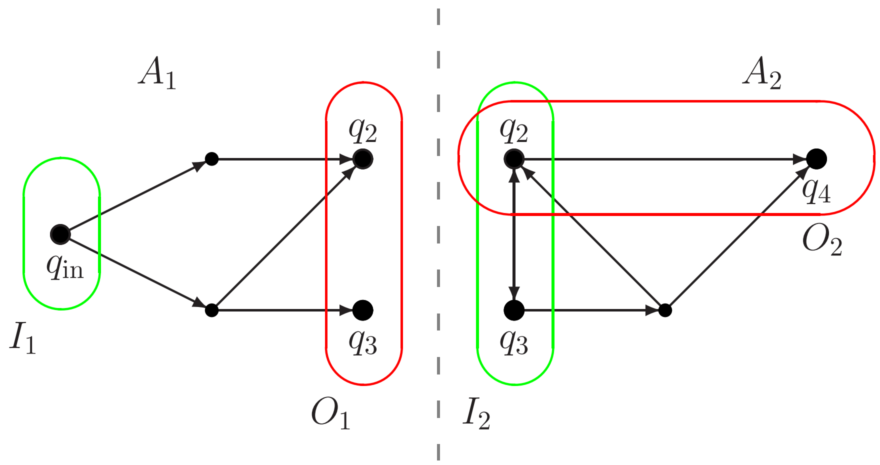

Fig. 1. Part of a lineage $\mathcal{A}$. The set of exit states of A_1 is a subset of the set of entry states of A_2.

A lineage $\mathcal{A}$ operates on elements of Σ^∞. On an input string x, at any time, only one automaton processes x. The automaton that processes x at a particular time is called *active* (at that time). Initially, A_1 is the active automaton, and it starts processing x. Whenever an active automaton A_i enters an exit state q, it turns the control over to A_{i+1}, which then becomes the active automaton. This is done by letting A_{i+1} start processing the remainder of x, 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 $\mathcal{A}$.

Formally, 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:

$$
\begin{aligned}
q_1 &:= q_{\text{in}} \ , \\
m_1 &:= 1 \ , \\
\\
q_{j+1} &:= \pi_1\left(\delta_{m_j}(q_j, x_j)\right) \ , \\
m_{j+1} &:= \begin{cases} m_j + 1 & \text{if } q_{j+1} \in O_{m_j} \\ m_j & \text{if } q_{j+1} \in Q_{m_j} - O_{m_j} \\ \text{undefined} & \text{if } q_{j+1} \text{ is undefined} \end{cases} \ .
\end{aligned}
\tag{3}
$$

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 emphasise the dependence. If q_j is defined for every $j \leq |x| + 1$, 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 $y_j := \pi_2\left(\delta_{m_j}(q_j, x_j)\right)$, for every $j \geq 1$.

Definition 3. *Let $\mathcal{A}$ be a lineage. For every $n \geq 1$, we define the partial function $\phi^{\mathcal{A},n} : \Sigma^n \to \Omega^n$ by letting $\phi^{\mathcal{A},n}(x)$ be the output of $\mathcal{A}$ on x if x is a valid input and undefined* *otherwise, for every x of length n. We say that $\phi^{\mathcal{A},n}$ is* realized *by the lineage $\mathcal{A}$. In general, for a partial function $\psi : \Sigma^n \to \Omega^n$, we say that ψ is* realizable, *if there is a lineage $\mathcal{A}$ such that ψ equals $\phi^{\mathcal{A},n}$.*

Since there are only finitely many strings of length n, for any lineage $\mathcal{A}$ and integer n, the translation $\phi^{\mathcal{A},n}$ can be realized by a single finite-state automaton. We conclude that $\phi^{\mathcal{A},n}$ is regular for all $\mathcal{A}$ and all n, which justifies the term realizable. There is no need to restrict our attention to finite strings.

Definition 4. *Let $\mathcal{A}$ be a lineage. We define the partial function $\Phi^{\mathcal{A}} : \Sigma^\infty \to \Omega^\infty$ 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 infinite string x. We say that $\Phi^{\mathcal{A}}$ is* non-uniform realized *by the lineage $\mathcal{A}$. In general, for a partial function $\Psi : \Sigma^\infty \to \Omega^\infty$, we say that Ψ is* non-uniform realizable, *if there is a lineage $\mathcal{A}$ such that Ψ equals $\Phi^{\mathcal{A}}$.*

It will become clear that for some lineages $\mathcal{A}$, the translation $\Phi^{\mathcal{A}}$ is not realizable by a single finite-state automaton (Prop. 5), which is why we use the term "non-uniform realization" to indicate realisability by a lineage of automata. For the remainder of this paper, we consider translations on infinite strings, unless stated otherwise. See also [6].

Let Φ be a translation and n an integer. We say that $(\Phi)_n$[1] *depends only on the first n input symbols*, if there is a function $f : \Sigma^n \to \Omega$, such that $(\Phi(x))_n$ equals $f(x_{[1:n]})$ for every x in the domain of Φ. A non-uniform realizable translation has this property for every n, since

$$(\Phi(x))_n = \pi_2\left(\delta_{m_n}(q_n, x_n)\right) = \pi_2\left(\delta_{m_n\left(x_{[1:n-1]}\right)}\left(q_n\left(x_{[1:n-1]}\right), x_n\right)\right) \quad . \tag{4}$$

Let $\mathcal{A} = A_1, A_2, A_3, \ldots$ be a lineage of automata and let n and m be integers. In a slight abuse of notation, we say that A_m *is able to process all strings of length n*, if $m_n(x_{[1:n]}) \leq m$ for every string x. In other words, if for any string x, the lineage $\mathcal{A}$ needs less than m updates to process the first n symbols of x.

3 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-uniform realize the same translation, i.e. $\Phi^{\mathcal{B}}$ equals $\Phi^{\mathcal{A}}$. In fact, we show two extreme cases: a method that postpones updates of automata to arbitrary finite time 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^A be the set of states, I^A the set of entry states and O^A the set of exit states of an automaton A.

3.1 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 that non-uniform realizes the same translation, we let $B_j := A_j$ for all $j < i$, and we let $B_j := A_{j+1}$ for all $j > i$.

[1] Where $(\Phi)_n$ is shorthand for the function $x \mapsto (\Phi(x))_n$.

Construction 1. Let Q^{B_i} be the disjoint union of Q^{A_i} and $Q^{A_{i+1}}$, that is,

$$Q^{B_i} = \{ \, (q,i) \mid q \in Q^{A_i} \, \} \; \cup \; \{ \, (q,i+1) \mid q \in Q^{A_{i+1}} \, \} \; . \tag{5}$$

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

$$\begin{aligned} I^{B_i} &= \{ \, (q,i) \mid q \in I^{A_i} \, \} \; , \\ O^{B_i} &= \{ \, (q,i+1) \mid q \in O^{A_{i+1}} - O^{A_i} \, \} \; \cup \; \{ \, (q,i) \mid q \in O^{A_{i+1}} \cap O^{A_i} \, \} \; . \end{aligned} \tag{6}$$

Let δ_i and δ_{i+1} be the transition functions of A_i and A_{i+1}, respectively. The transition function γ_i of B_i is defined for all states q and all $a \in \Sigma$, 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 $\gamma_i((q, i), a)$.

- 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).

In order to achieve continuity, we should relabel every state q of B_j as (q, i) for $j < i$, and $(q, i+1)$ for $j > i$ (unless q is an exit state of both A_i and A_{i+1}, in which case (q, i) is the correct label), and adjust the transitions accordingly.

See Fig. 2 for an example.

Proposition 1. *Lineage $\mathcal{B}$ from Construction 1 non-uniform realizes the same translation as $\mathcal{A}$.*

Proof. Since the states of B_j have been relabelled 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

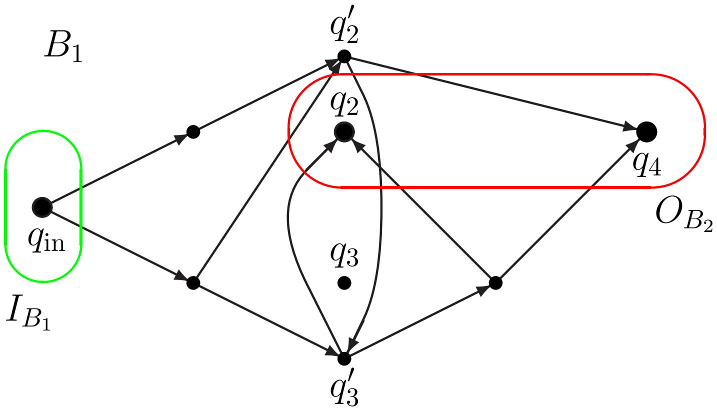

Fig. 2. The automata A_1 and A_2 from Fig. 1 are replaced by the automaton B_1.

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 relabelled 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 i or $i+1$ resp. appended to the state). It follows that $\mathcal{B}$ non-uniform realizes the same translation as $\mathcal{A}$.

□

This construction can be applied repeatedly to merge any fixed number of automata. The intuitive notion behind lineages is that they model the evolution of a system. Applying Construction 1 on a lineage can be thought of as "making a bigger jump in the evolution of the system".

3.2 Updating the Lineage at Each Step

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

Construction 2. First, we let the set of states for the lineage $\mathcal{B}$ be

$$\{\ (q,i)\quad|\quad q \in A_i\ \}\ . \tag{7}$$

Now, we recursively construct the automaton B_n. Let q_{in} be the initial state of A_1. Then the initial state of B_1 is the state $(q_{\text{in}}, 1)$, and $I^{B_1} = \{(q_{\text{in}}, 1)\}$. 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 δ_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 $\gamma_n((q,i),a) = ((r,i'),b)$, where γ_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 2. *Lineage $\mathcal{B}$ from Construction 2 non-uniform realizes the same translation as $\mathcal{A}$.*

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 $\mathcal{B}$ non-uniform realizes the same translation as $\mathcal{A}$.

□

Applying Construction 2 on a lineage can be thought of as "taking smaller steps in the evolution of the system".

3.3 Reducing the Number of States

The next method works only on lineages that update after every input symbol. From section 3.2, 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 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 $\mathcal{B}$ 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 δ_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 γ_n is the transition function of B_n. Since f_n and g_n are injective, this definition is unambiguous.

Proposition 3. *Lineage $\mathcal{B}$ from Construction 3 non-uniform realizes the same translation as $\mathcal{A}$.*

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 like the previous method, we can conclude from this fact that $\mathcal{B}$ non-uniform realizes the same translation as $\mathcal{A}$.

□

We summarise the last two constructions in the following result:

Proposition 4. *Every non-uniform realizable translation can be non-uniform realized by a lineage $\mathcal{B}$ with the property that its automata B_i update after every step and have precisely $\max\{|I^{B_i}|, |O^{B_i}|\}$ states each.*

4 Properties of the Class of Non-Uniform Realizable Translations

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

Proposition 5. *Let Ω be an alphabet with at least two elements. There are uncountably many non-uniform 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. It is left to the reader to construct a lineage that non-uniform realizes the translation Φ, defined by

$$\Phi(x) = 1^{n_1}01^{n_2}01^{n_3}0\dots \quad , \tag{8}$$

for every x in Σ^ω. Suppose that Φ can be realized by an automaton A. Let a be a letter in Σ and let π be a program that, on input $i \in \mathbb{N}$ in unary, simulates A on input a^ω until A has written i zeroes. Then π outputs the last sequence of ones in A's output. We see that π computes n_i in unary. It follows that π enumerates N, which is a contradiction. Because there are uncountably many sets that are not recursively enumerable, we have the desired result. □

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

4.1 A Lineage-Independent Characterisation of Non-Uniform Realizable Translations

It is possible to characterise the translations that are non-uniform realized by a lineage without actually constructing lineages, by specialising the theory of continuous mappings ([8]). The characterisation depends on the domain on which the translation is defined and the relation between input and output pairs.

Theorem 1. *A translation Φ can be non-uniform realized by a lineage $\mathcal{A}$ iff the domain of Φ is closed and $(\Phi)_n$ depends only on the first n input symbols, for every n.*

We prove it with the following two lemmas.

Lemma 1. *Let Φ be a translation. Suppose the domain of Φ is closed and $(\Phi)_n$ depends only on the first n input symbols, for every n. Then Φ is non-uniform realizable.*

Proof. Let D be the domain of Φ. Let f_n be the function with domain $P^n(D)$, such that $(\Phi(x))_n = f_n(x_{[1:n]})$ for every $x \in D$. By the assumptions of the Lemma, f_n is well-defined. We construct a lineage $\mathcal{A}$ that non-uniform realizes Φ. Every state of $\mathcal{A}$ will correspond to a prefix of a string in D.

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

$$\begin{aligned} I_n &= \{ \ [\![u]\!] \quad | \quad u \in P^{n-1}(D) \ \} \ , \\ O_n &= \{ \ [\![u]\!] \quad | \quad u \in P^{n}(D) \ \} \ , \end{aligned} \tag{9}$$

and $Q_n = I_n \cup O_n$. The initial state of A_1 is $[\![\epsilon]\!]$. The transition function δ_n is defined by

$$\delta_n([\![u]\!], a) := \begin{cases} ([\![ua]\!], f_n(ua)) & \text{if } ua \in P^n(D) \\ \text{undefined} & \text{otherwise} \end{cases} .$$

To see that $\mathcal{A}$ non-uniform realizes Φ, let $x \in D$ be an input to $\mathcal{A}$. Using induction, we claim that in the n-th update, $\mathcal{A}$ enters state $[\![x_{[1:n]}]\!]$ with output $f_n(x_{[1:n]})$.

- For $n = 1$, the lineage starts in $[\![\epsilon]\!]$. The transition $\delta_1([\![\epsilon]\!], x_1)$ is defined because $|\epsilon| = 0$ and ϵx_1 is a prefix of $x \in D$, so A_1 enters state $[\![x_1]\!]$ with output $f_1(x_1)$.
- Assume the claim holds for a certain n. The lineage is updated after reading x_n, so A_{n+1} begins in state $[\![x_{[1:n]}]\!]$. By the same argument, A_{n+1} enters state $[\![x_{[1:n+1]}]\!]$ with output $f_{n+1}(x_{[1:n+1]})$.

So the claim holds for every n. By the definition of f_n, we see that $\Phi^{\mathcal{A}}(x) = \Phi(x)$ for every $x \in D$.

Suppose on the other hand that $x \notin D$. Let D^c be the complement of D. Since D^c is open, there is a basis set $\mathbb{B}(u)$ that contains x, which is a subset of D^c. It follows that every element of $\mathbb{B}(u)$ is not in D, and so $u \notin P^{|u|}(D)$. Since the transition functions are not defined on u, we see that x is not a valid input to $\mathcal{A}$. Hence $\mathcal{A}$ non-uniform realizes Φ.

□

Lemma 2. *Let Φ be a non-uniform realizable translation. Then the domain of Φ is closed and $\pi_n \circ \Phi$ depends only on the first n input symbols, for every n.*

Proof. Let D be the domain of Φ, let D^c the complement of D and let $\mathcal{A}$ be a lineage that non-uniform realizes Φ. Let $x \in D^c$ be an infinite string and consider a run of $\mathcal{A}$ on x. Because x is not in the domain of Φ, it must be the case that at some point, after processing a prefix of length $n-1$ of x, the automaton that is active at that time, say A_k, is in a certain state q, such that there is no transition from q with x_n as input. If this moment would not occur, then $\mathcal{A}$ would never halt during the run, and x would be in the domain.

Let y be a string in $\mathbb{B}(x_{[1:n]})$ and consider a run of $\mathcal{A}$ on y. Since the first n symbols of x and y are the same, the computation will halt at the same point during the computation, so y cannot be in the domain of Φ. Hence $\mathbb{B}(x_{[1:n]})$ is a subset of D^c, which implies that D^c is an open set, and D is closed.

It follows from (4), that $(\Phi(x))_n$ depends only on the first n input symbols.

□

For an example of a translation that cannot be non-uniform realized by a lineage, consider the translation Φ^{-1} from Example 1 in the sequel.

4.2 Composition of Non-Uniform 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-uniform realizable translations is also closed under these operations. This question is answered in this and the next subsection.

Proposition 6. *Let $\Phi^{\mathcal{A}} : \Sigma \to \Omega$ and $\Phi^{\mathcal{B}} : \Omega \to \Theta$ be translations non-uniform realized by lineages $\mathcal{A}$ and $\mathcal{B}$. Then a lineage $\mathcal{C}$ exists such that $\Phi^{\mathcal{C}} = \Phi^{\mathcal{B}} \circ \Phi^{\mathcal{A}}$.*

Proof. We construct a lineage $\mathcal{C}$, by defining for every automaton C_i its set of states, its initial state and its transition function. Define the set of states of C_i by:

$$Q^{C_i} = \{ \ (q, k, r, l) \quad | \quad k, l \leq i + 1, \ q \in Q^{A_k} \wedge r \in Q^{B_l} \ \} \ . \tag{10}$$

When C_i is in state (q, k, r, l), this simulates the fact that A_k is active in state q and B_l is active in state r. So $\mathcal{C}$ simulates the transitions of $\mathcal{A}$ in the first two components of its states, and the transitions of $\mathcal{B}$, with the output of $\mathcal{A}$ as input, in the last two components.

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 $(q_{\text{in}}, 1, r_{\text{in}}, 1)$. A state (q, k, r, l) is an exit state if q is an exit state of A_{k-1} (and $k > 1$)[2] 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 μ_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 δ_k be the transition function of A_k, and γ_l the transition function of B_l, and suppose that $\delta_k(q, a) = (q', b)$, and $\gamma_l(r, b) = (r', c)$. If $\mathcal{A}$ is in the k'-th update[3] and $\mathcal{B}$ is in the l'-th update *after* reading a and b, respectively, then we let $\beta_i((q, k, r, l), a) = ((q', k', r', l'), c)$. In all other cases we let μ_i be undefined. It follows that $\mathcal{C}$ updates whenever $\mathcal{A}$ or $\mathcal{B}$ updates.

Proving that $\mathcal{C}$ non-uniform realizes the translation $\Phi^{\mathcal{B}} \circ \Phi^{\mathcal{A}}$ is similar to the proof of Prop. 2. If x is an input to $\mathcal{A}$ and $y = \Phi^{\mathcal{A}}(x)$ is an input to $\mathcal{B}$ 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 $\Phi^{\mathcal{B}}(\Phi^{\mathcal{A}}(x)) = \Phi^{\mathcal{C}}(x)$. If, on the other hand, either x or y is not valid for $\mathcal{A}$ or $\mathcal{B}$ respectively, then there comes a time when either $\delta_k(q, x_n)$ or $\gamma_l(r, y_n)$ is undefined. In either case, $\mu_i((q, k, r, l), x_n)$ is also undefined. Hence the domains match and the two functions are equal.

□

The set of states Q_{C_i} in the given proof can be taken to be 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.

Let the complexity of a lineage $\mathcal{A}$ be measured by a function g such that the n-th automaton of $\mathcal{A}$ has $g(n)$ states (see section 5 for more details). Then we can express the complexity of the lineage $\mathcal{C}$ from Prop. 6 in terms of the complexities of $\mathcal{A}$ and $\mathcal{B}$. Let $g^{\mathcal{A}}$ be the complexity of $\mathcal{A}$ and $g^{\mathcal{B}}$ the complexity of $\mathcal{B}$. Then the complexity of $\mathcal{C}$ is

$$g^{\mathcal{C}}(n) = \sum_{i,j \leq n+1} g^{\mathcal{A}}(i) \cdot g^{\mathcal{B}}(j) \ . \tag{11}$$

Corollary 1. *The composition of two lineages with polynomial complexity has polynomial complexity too.*

---

[2] Right after A_{k-1} enters q, an update takes place, so the corresponding state is $(q, k, \_, \_)$ instead of $(q, k - 1, \_, \_)$. The same holds for r in B_{l-1}.

[3] Explicitly: if q' is an exit state of A_k, then $k' := k + 1$, else $k' := k$. The same holds for l'.

4.3 The Inverse of a Non-Uniform Realizable Translation

It is not the case that every non-uniform realizable injective translation has a non-uniform realizable inverse, see Example 1.

Example 1. Let $\Sigma = \{a, b, c, d\}$, and $\Omega = \{0, 1\}$. Encode the symbols of Σ in Ω as follows:

$$\begin{array}{ll} \mathrm{enc}(a) = 00 \ , & \mathrm{enc}(c) = 10 \ , \\ \mathrm{enc}(b) = 01 \ , & \mathrm{enc}(d) = 11 \ . \end{array} \tag{12}$$

Define the translation $\Phi : \Sigma^\infty \to \Omega^\infty$ by

$$\Phi(x) = \mathrm{enc}(x_1)\mathrm{enc}(x_2)\mathrm{enc}(x_3)\ldots \ . \tag{13}$$

Clearly, $(\Phi)_n$ depends only on the $\lceil n/2 \rceil$-th input symbol, so Φ is non-uniform realizable by Lemma 1. It is also not hard to see that Φ is bijective.

To see that Φ^{-1} cannot be non-uniform realized, consider the strings 0^ω and 01^ω. Now $\Phi^{-1}(0^\omega) = a^\omega$ and $\Phi^{-1}(01^\omega) = bd^\omega$. Suppose $\mathcal{A}$ is a lineage that non-uniform realizes Φ^{-1}. Given 0 as the first input symbol, A_1 must decide which symbol to output. If A_1 gives a, then $\mathcal{A}$ fails to process 01^ω correctly, but if it doesn't give a, then 0^ω cannot be properly processed. Thus $\mathcal{A}$ does not non-uniform realize Φ^{-1}.

Fortunately, we can give a necessary and sufficient condition for injective translations to have a non-uniform realizable inverse. First, we give a sufficient condition.

Proposition 7. *Let Φ be a translation with domain D, non-uniform realized by a lineage $\mathcal{A}$. Consider the functions $\phi^{\mathcal{A},n}$. If $\phi^{\mathcal{A},n}$ is injective for every n, then the translation Φ^{-1} with domain $\Phi(D)$ can be non-uniform realized.*

Proof. We will transform $\mathcal{A}$ into a lineage $\mathcal{B}$ that non-uniform realizes Φ^{-1}. Consider the automaton A_k, with transition function δ_k. First, we remove all states that cannot be reached from the initial state. Let q be a state and a a letter. If $\delta_k(q, a) = (r, b)$, then we define $\gamma_k(q, b) = (r, a)$.

We claim that γ_k is well-defined. To prove the claim, suppose that $\delta_k(q, a) = (r, b)$ and $\delta_k(q, a') = (r', b)$, for some a and a' with $a \neq a'$. Let u be a prefix 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, but $ua \neq ua'$. Hence the function $\phi^{\mathcal{A},|u|+1}$ is not injective, which contradicts the condition of the Proposition. It follows that γ_k is well-defined.

The automaton B_k is defined by taking A_k, with δ_k replaced by γ_k. Let $x \in D$ be an input to Φ with output $y = \Phi(x)$. For every n, let m_n be the index of the automaton that is active at time n, and let q_n be the state that A_{m_n} is in at that time. Suppose y is input to $\mathcal{B}$. It is left to the reader to show that $\mathcal{B}$ is in its m_{n+1}-st automaton in state q_{n+1}, after processing $y_{[1:n]}$, for every n. It follows that the output of $\mathcal{B}$ is $x = \Phi^{-1}(y)$.

Let y be a string not in $\Phi(D)$. Let v be the largest prefix of y that $\mathcal{A}$ can output, with some suitable finite input. Let u be the input that causes $\mathcal{A}$ to output v. Suppose $\mathcal{A}$ is in its k-th update in state q after processing u. We see that there is no transition from q that gives $y_{|v|+1}$ as output (otherwise v would not be the largest prefix). It follows that there is no transition from q with input $y_{|v|+1}$ in B_k. We conclude that y is not a valid input to $\mathcal{B}$, since the first $|v|$ symbols of y cause $\mathcal{B}$ to enter q, but there is no transition from q with input $y_{|v|+1}$. Hence the domain of $\mathcal{B}$ is $\Phi(D)$. We conclude that $\mathcal{B}$ non-uniform realizes Φ^{-1}.

□

The lineage that is constructed in the proof of Prop. 7 has the same complexity as the lineage that we started with (after removing the redundant states). It follows that any two non-uniform realizable translations that are each others inverse belong to the same complexity class.

Next, we show that the condition of Prop. 7 is also necessary for translations to have a non-uniform realizable inverse.

Proposition 8. *Let Φ be a non-uniform realizable injective translation with domain D. Suppose Φ^{-1}, with domain $\Phi(D)$, is also non-uniform realizable. Then there is a lineage $\mathcal{A}$ that non-uniform realizes Φ, such that $\phi^{\mathcal{A},n}$ is injective for every n.*

Proof. Let $R = \Phi(D)$. Let $\mathcal{B}$ be a lineage that non-uniform realizes Φ, and $\mathcal{C}$ a lineage that non-uniform realizes Φ^{-1}. Fix an n and let ϕ be the restriction of $\phi^{\mathcal{B},n}$ to $P^n(D)$, and ψ the restriction of $\phi^{\mathcal{C},n}$ to $P^n(R)$.

Suppose $\phi(u) = \phi(u') = v$. Let $x \in D$ be an infinite extension of u, and let $y = \Phi(x)$. Then $\Phi^{-1}(y) = x$, and since Φ^{-1} is non-uniform realizable, it follows that $\psi(v) = u$. Using a similar argument, we also show that $\psi(v) = u'$. We conclude that u must equal u', so ϕ is injective. Using Lemma 1, we can construct a lineage $\mathcal{A}$ such that $\phi^{\mathcal{A},n} = \phi$.

□

We conclude from Prop. 7 and 8 that a non-uniform realizable injective translation Φ has a non-uniform realizable inverse iff the function $(\Phi)_{[1:n]}$ is injective for every n. If the input alphabet and the output alphabet have the same size, then any non-uniform realizable bijective translation has a non-uniform realizable inverse.

Theorem 2. *Let $\Phi : \Sigma \to \Omega$ be a non-uniform realizable translation, with $|\Sigma| = |\Omega|$. If Φ is a bijection, then Φ^{-1} is also non-uniform realizable.*

Proof. Let $\mathcal{A}$ be a lineage that non-uniform realizes Φ, and let $\phi^{\mathcal{A},n}$ be the corresponding restriction to strings of length n. Since Φ is surjective, $\phi^{\mathcal{A},n}$ must also be surjective. But this implies that $\phi^{\mathcal{A},n}$ is injective. Now we apply Prop. 7.

□

To see that $|\Sigma| = |\Omega|$ is really necessary for bijections, assume that $|\Sigma| < |\Omega|$. Since Φ is non-uniform realizable, $\pi_1 \circ \Phi$ depends only on the first input symbol. It follows that there are only $|\Sigma|$ choices for $\pi_1 \circ \Phi$, which implies that Φ is not surjective. Hence Φ cannot be both non-uniform realizable and bijective at the same time, unless $|\Sigma| \geq |\Omega|$. If $|\Sigma| > |\Omega|$, then a similar argument shows that Φ^{-1} cannot be non-uniform realizable and bijective at once, see also Example 1. Alternatively, we can use Prop. 8 to see that both $\pi_1 \circ \Phi$ and $\pi_1 \circ \Phi^{-1}$ have to be injective, and since they are both total functions, $|\Sigma|$ must equal $|\Omega|$.

5 The Complexity of Lineages

We have already mentioned the notion of complexity of lineages. In this section, we develop the concept further and establish several results which allow us to compare many translations, based on the complexity of the lineages that non-uniform realize them.

The processing power of an automaton is directly related to the number of states. An automaton with more states is able to distinguish among a greater number of different situations. It can apply different actions to each situation it can recognise, thus adding more diversity to a computation.

For a lineage, which is a sequence of automata, the number of states of each of the constituent automata contributes to the computing power of the lineage. Therefore, we use a function to describe the complexity of a lineage.

Definition 5. *The* complexity *of a lineage $\mathcal{A}$ is a function g such that for every n, the number of states of A_n equals $g(n)$. We say that a translation Φ is of complexity g if there is a lineage $\mathcal{A}$ of complexity g that non-uniform realizes Φ. We define the complexity class SIZE(g) as the class of non-uniform realizable translations of complexity g or less.*

First, we give an upper bound on the complexity of any non-uniform realizable translation.

Proposition 9. *Let Φ be a non-uniform realizable translation over an alphabet of size c. Then Φ can be non-uniform realized by a lineage of size at most c^n.*

Proof. Let $\mathcal{A}$ be a lineage that non-uniform realizes Φ. We apply Prop. 4 on $\mathcal{A}$, to obtain a lineage $\mathcal{B}$ that non-uniform realizes the same translation. Let's look at the size complexity of $\mathcal{B}$. From the initial state, at most c different states can be reached in one step. Thus B_1 has at most c states. Suppose B_n has $m \leq c^n$ states. Then B_{n+1} has at most m entry states, so at most $c \cdot m$ states can be reached in one step. It follows that B_{n+1} has at most $c \cdot m$ states, with is at most c^{n+1}. We see that the complexity of $\mathcal{B}$ is at most c^n.

□

5.1 Complexity Classes and the Functions that Represent Them

We have expressed the complexity of an evolving interactive system by a positive integer-valued function. Conversely, we ask which positive integer-valued functions represent a complexity class. Some functions do not naturally correspond to a complexity class, e.g. the super-exponential functions (Prop. 9). If a function is non-decreasing and has a "growth rate[4] "that is bounded by a constant, then it corresponds to a complexity class. If it is not, then we take a suitable function that is nowhere greater than the original function and consider its corresponding class. This is made precise below.

Let $g : \mathbb{N} \to \mathbb{N}$ be an arbitrary positive non-decreasing function and c a positive integer. Define the function $g_c(n)$ by

$$\begin{aligned} g_c(1) &= \min\{\ g(1)\ ,\ c\ \}\ , \\ g_c(n+1) &= \min\{\ g(n+1)\ ,\ c \cdot g_c(n)\ \}\ . \end{aligned} \tag{14}$$

It follows that for every n,

- $g_c(n) \leq g(n)$,
- $g_c(n) \leq c^n$,
- $g_c(n) \leq g_c(n+1)$, and
- $g_c(n+1) \leq c \cdot g_c(n)$.

In other words, g_c is a positive, non-decreasing function that is bounded by g and c^n, with a "growth rate" that is bounded by c.

To show that the class SIZE(g_c) is not empty, we construct a translation $\Phi^{g,c}$ that is in this class. We also show that $\Phi^{g,c}$ is not in SIZE(h), for any function h such that $h(n) < g_c(n)$

[4] The growth rate of a function g is defined as $g(n+1)/g(n)$

for some n. The construction is based on the following idea: suppose $\mathcal{A}$ is a lineage that non-uniform realizes a translation Φ. Two input-prefixes u and v (not necessarily of the same size) are considered inequivalent when there is an infinite string x and an integer i such that $\Phi(ux)$ and $\Phi(vx)$ both exist and $(\Phi(ux))_{|u|+i} \neq (\Phi(vx))_{|v|+i}$. This is impossible if $\mathcal{A}$ enters the same state after processing either u or v. Thus, to make sure that an automaton of a lineage has at least k states, we need to make sure that there are at least k inequivalent inputs to choose from, once this automaton is active.

First, we establish the domain of $\Phi^{g,c}$. Let Σ be an alphabet of size c. For every n, we choose $g_c(n)$ strings of length n, such that they are prefixes of the $g_c(n+1)$ strings of length $n+1$. The details are given in Construction 4.

Construction 4. Label the letters from Σ as a_1 through a_c, and let C_n be the chosen subset of size $g_c(n)$ of Σ^n. We proceed recursively.

$$C_1 = \{\ a_i \quad | \quad i \leq g_c(1)\ \}\ . \tag{15}$$

Assume C_n is chosen. Using integer division, we write $g_c(n+1) = l \cdot g_c(n) + m$, for unique integers l and m, with $0 \leq m < g_c(n)$. It follows that $1 \leq l \leq c$. Let $u_1, \ldots, u_m$ be m different strings in C_n. Now take

$$C_{n+1} = \{\ ua_i \quad | \quad u \in C_n \ \wedge\ i \leq l\ \} \cup \{\ u_j a_{l+1} \quad | \quad j \leq m\ \}\ . \tag{16}$$

It is left to the reader to verify that C_{n+1} contains $g_c(n+1)$ strings of length $n+1$. Note that C_{n+1} is well-defined, as either $l < c$ or $l = c$ and $m = 0$.

We call a string in C_n a *choice*. Note that ua_1 is a choice if u is a choice. Consider an infinite sequence of choices, such that each choice is a prefix of its successor. Such a sequence defines a unique infinite string, such that each of its prefixes is a choice. Let Δ be the set of infinite strings x such that each prefix of x is a choice. The translation $\Phi^{g,c}$ will be defined on the domain Δ.

Construction 5. Consider the family of functions $f_{k,l,m} : \Sigma^\infty \to \Sigma^{1+l}$, defined by

$$f_{k,l,m}(x) = \begin{cases} x_k^{1+l} & \text{if } 0 < k \leq m \\ x_1^{1+l} & \text{otherwise} \end{cases}\ , \tag{17}$$

for $k, l, m \in \mathbb{N}$. Let $\psi : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ be a surjective function that attains each value infinitely often. Then the translation Φ is defined by

$$\Phi(x) = f_{\psi(1),1}(x) f_{\psi(2),2}(x) f_{\psi(3),3}(x) \ldots \tag{18}$$

Finally, we define the translation $\Phi^{g,c}$ by

$$\Phi^{g,c}(x) = \begin{cases} \Phi(x) & \text{if } x \in \Delta \\ \text{undefined} & \text{otherwise} \end{cases}\ . \tag{19}$$

The translation $\Phi^{g,c}$ can be non-uniform realized by a lineage of automata. To prove this, we need the following two Lemmas.

Lemma 3. *Δ is a closed set.*

Proof. Let Δ^c be the complement of Δ. Suppose $x \in \Delta^c$. Then there is a prefix u of x such that u is not a choice. Let y be an infinite string in $\mathbb{B}(u)$. Since u is a prefix of y, it follows that $y \in \Delta^c$. We conclude that Δ^c is an open set, so Δ is closed.

□

Lemma 4. *For every n, the function $\pi_n \circ \Phi^{g,c}$ depends only on the first n input symbols.*

Proof. Let $x \in \Delta$ be an infinite string. The output of $\Phi(x)$ consists of infinitely many concatenations of strings of the form $f_{\psi(m),m}(x)$. For every integer $m \geq 1$, the string $f_{\psi(m),m}(x)$ starts at index

$$i_m = \left(\sum_{t=1}^{m-1} |f_{\psi(t),t}(x)| \right) + 1 \geq m \quad . \tag{20}$$

This string consists of multiple copies of x_k, for a certain $k \leq m$. Note that k does not depend on the particular choice of x. The n-th symbol of $\Phi(x)$ belongs to $f_{\psi(m),m}(x)$ for a certain m. Obviously $n \geq i_m$, so $k \leq m \leq i_m \leq n$. It follows that $\pi_n(\Phi^{g,c}(x)) = x_k$ for a certain $k \leq n$.

□

Combining Lemmas 3, 4 and 1, we conclude that $\Phi^{g,c}$ can be non-uniform realized. Next, we will examine the complexity of $\Phi^{g,c}$. Prop. 10 shows that $\Phi^{g,c}$ is of complexity g_c, while Prop. 11 tells us that any lineage with a complexity less than g_c cannot non-uniform realize $\Phi^{g,c}$.

Proposition 10. *The translation $\Phi^{g,c}$ can be non-uniform realized by a lineage $\mathcal{A}$ that updates at every step, such that A_n has $g_c(n)$ states.*

Proof. Let $\mathcal{B}$ be the lineage from the proof of Lemma 1. We see that $P^n(\Delta)$ equals C_n. It follows that the number of entry states of B_n is $g_c(n-1)$, and the number of exit states of B_n is $g_c(n)$. Remember that $g_c(n-1) \leq g_c(n)$. Using Construction 3, we can transform $\mathcal{B}$ into a lineage $\mathcal{A}$ with $g_c(n)$ states, that updates at every step.

□

For the proof of Prop. 11, we need the following Lemma.

Lemma 5. *Let k, l and $n \geq 1$ be integers such that $k, l \geq n$. Let x and y be infinite strings such that $x_n \neq y_n$. Then there is an i such that*

$$\pi_{k+i}(\Phi(x)) \neq \pi_{l+i}(\Phi(y)) \quad . \tag{21}$$

Proof. Assume $k \geq l$. Let $t = k - l$. Choose an integer $m \geq l$ such that $\psi(m) = (n, t)$. Then $f_{\psi(m),m}(x) = f_{n,t,m}(x) = x_n^{1+t}$, since $n \leq m$. It follows that $\Phi(x)$ contains a string x_n^{1+t}, starting at an index $i_m \geq m$, namely the string $f_{\psi(m),m}(x)$. Similarly, $\Phi(y)$ contains a string y_n^{1+t}, starting at the same index, see Fig. 3. But then

$$\pi_{i_m+t}(\Phi(x)) \neq \pi_{i_m}(\Phi(y)) \quad , \tag{22}$$

since $x_n \neq y_n$. Now $i_m \geq m \geq l$, so $i_m = l+i$ for a certain i, and $i_m+t = l+i+(k-l) = k+i$. Therefore

$$\pi_{k+i}(\Phi(x)) \neq \pi_{l+i}(\Phi(y)) \quad . \tag{23}$$

□

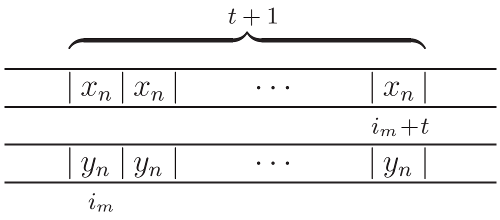

Fig. 3. Part of the outputs of Φ, on the inputs x and y. Starting in position i_m, the outputs contain a sequence of x_n's and y_n's respectively, of size $t+1$ each. As a result, $\pi_{i_m+t}(\Phi(x)) = x_n \neq y_n = \pi_{i_m}(\Phi(y))$.

Let u and v be two different choices of length n. Let u' be a choice that extends u and v' a choice that extends v. Finally, let $x = (a_1)^\omega$. It follows that $u'x$ and $v'x$ are elements of Δ. Since $u \neq v$, it follows that there is an $n' \leq n$, such that $\pi_{n'}(u'x) \neq \pi_{n'}(v'x)$. By Lemma 5, there is an i such that

$$\pi_{|u'|+i}(\Phi^{g,c}(u'x)) \neq \pi_{|v'|+i}(\Phi^{g,c}(v'x)) \ . \tag{24}$$

See Fig. 4 for a visual explanation.

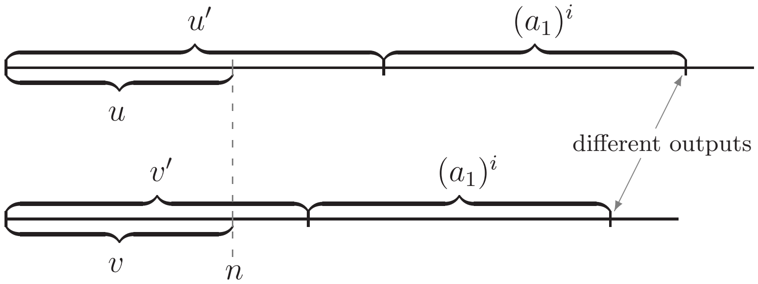

Fig. 4. Two finite choices u and v, such that $u \neq v$, are extended to choices u' and v' respectively. These choices are extended with the infinite string $x = (a_1)^\omega$. Then there is an integer i such that $\pi_{|u'|+i}(\Phi^{g,c}(u'x)) \neq \pi_{|v'|+i}(\Phi^{g,c}(v'x))$.

Proposition 11. *Let $\mathcal{A}$ be a lineage that non-uniform realizes $\Phi^{g,c}$. Suppose A_m is able to process all strings of length n. Then A_m has at least $g_c(n)$ states.*

Proof. Suppose A_m has less than $g_c(n)$ states. Then there are two different choices u and v of length n, with choices u' and v' that extend u and v respectively, such that A_m enters the same state r after processing either u' or v', see Fig. 5.

Then there is an i that satisfies (24). Suppose $\mathcal{A}$ is in the m-th update[5], in state r. Now we give $x = (a_1)^\omega$ as further input to $\mathcal{A}$. After i steps, $\mathcal{A}$ enters a state r' with a certain output b. These last i steps are independent of the steps that $\mathcal{A}$ took to reach r. In other words,

$$\pi_{|u'|+i}(\Phi^{g,c}(u'x)) = \pi_{|v'|+i}(\Phi^{g,c}(v'x)) = b \ , \tag{25}$$

[5] Or the $m+1$-st, if r is an exit state.

which contradicts (24). It follows that A_m must have at least $g_c(n)$ states.

□

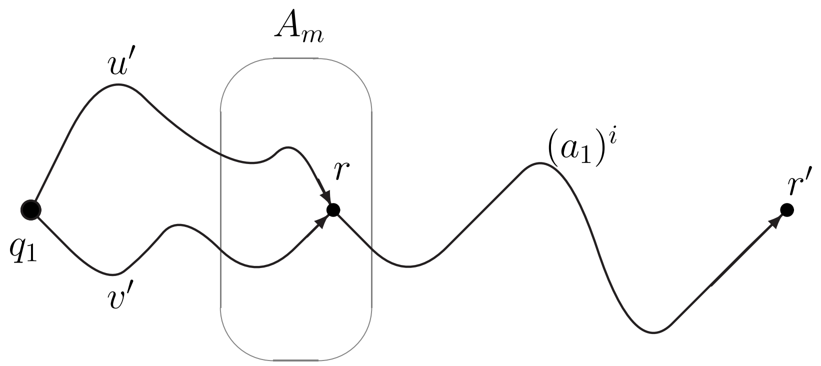

Fig. 5. The paths of two valid input-prefixes u' and v'. After processing u' or v', A_m enters the state r. Then the remainder of the input is processed, which equals $(a_1)^\omega$ in both cases. The rest of the path only depends on r and $(a_1)^\omega$, so after i steps, both paths enter r' and the same output symbol is generated.

Corollary 2. *Let $\mathcal{A}$ be a lineage that non-uniform realizes $\Phi^{g,c}$. Then A_n has at least $g_c(n)$ states.*

Proof. Since each active automaton must read at least one symbol before $\mathcal{A}$ can update, by the time $\mathcal{A}$ is ready to update to the $n+1$-st automaton, at least n symbols have been read.

□

For any non-decreasing function g and any integer $c > 1$, the complexity class SIZE(g_c) contains the translation $\Phi^{g,c}$. Furthermore, SIZE(g_c) is the smallest complexity class that contains $\Phi^{g,c}$.

5.2 A Hierarchy Result for Complexity Classes

For clarity, we repeat the results of the preceding section in one Proposition.

Proposition 12. *Let c be a positive integer and g a positive, non-decreasing function such that g equals g_c. Let h be a function such that $h(m) < g(m)$ for a certain m. Then SIZE(g) − SIZE(h) is non-empty.*

Proof. Combine Prop. 10 and Corollary 2.

□

When we are free to choose c, we can show that for any positive non-decreasing function g, translations exist that cannot be non-uniform realized by any lineage that has less than $g(n)$ states in its n-th automaton, for any n. Observe that this is a stronger claim then before; we no longer require the “growth rate” to be bounded.

Theorem 3. *Let g be a positive, non-decreasing function and let h be a function such that $h(m) < g(m)$ for a certain m. Then SIZE(g) − SIZE(h) is non-empty.*

Proof. Let $c \geq g(1)$ be an integer such that $g(n+1) \leq c \cdot g(n)$ for every n smaller than m. Then $g_c(n) = g(n)$ for all $n \leq m$. It follows that $h(m) < g_c(m)$. The translation $\Phi^{g,c}$ can be non-uniform realized by a lineage of size g_c (Prop. 10). Hence $\Phi^{g,c} \in$ SIZE(g).

Any lineage $\mathcal{A}$ that non-uniform realizes $\Phi^{g,c}$ must have at least $g_c(n)$ states in its n-th automaton (Corollary 2). Since $h(m) < g_c(m)$, it follows that $\mathcal{A}$ is not of size h. Hence $\Phi^{g,c} \notin$ SIZE(h).

□

Corollary 3. *Let g and h be positive non-decreasing functions such that $h(n) \leq g(n)$ for all n. If the inequality is strict for a certain n, then then SIZE(h) is a proper subset of SIZE(g).*

Proof. By definition, any translation of complexity h is in SIZE(g). By Thm. 3, not every translation of complexity g is in SIZE(h).

□

This means that every extra state of a lineage can be used to gain more potential computing power.

Corollary 4. *Let g and h be positive non-decreasing functions such that $h(n) < g(n)$ for a certain n and $g(m) < h(m)$ for a certain m. Then the classes SIZE(g) and SIZE(h) are incomparable, both contain translations that do not occur in the other.*

6 Relating Lineages to Interactive Turing Machines with Advice

Another model for evolving interactive computations is the interactive Turing machine with advice (ITM/A), introduced in [4, 6]. An ITM/A is an extension of an (ω-) Turing machine, which operates in an on-line fashion as a translator, taking infinite strings over an alphabet Σ as input and producing infinite output strings over an alphabet Ω. Additionally, the symbol λ can be used when no useful in- or output symbol is required. At any time, an ITM/A can enter a finite internal phase to do some internal computations. During such a phase, no in- or output occurs. An ITM/A also has an advice mechanism. At time t, it can call its advice only for values $t' \leq t$.

We say that an ITM/A realizes a translation Φ from Σ^∞ to Ω^∞, by letting $\Phi(x)$ be the output of the ITM/A, *without the gaps from the internal phases*, when given x as input. To this end, all λ's are removed from the actual in- and output. For more details, see [6].

Formally, we designate a subset of the control states of the ITM/A as *internal states*. The complement is referred to as the set of *external states*.

Definition 6. *A configuration with an internal state is an* internal configuration *and a configuration with an external state an* external configuration*. An* internal phase *is a maximal part of a computation that consists of only internal configurations.*

The transition function of an ITM/A is defined as usual, with the following restrictions: consider the transition $\delta(q, a) = (r, b)$.

– If $a = \lambda$, then $r = q$ and $b = \lambda$. The input symbol is ignored and the configuration remains unchanged.

- If q is external and r is internal, then $b = \lambda$. An internal phase starts, so no output is generated.
- If q and r are both internal, then $a = b = \lambda$. The ITM/A is in an internal phase, so the input symbol is ignored[6] and no output is generated.
- If q is internal and r is external, then $a = \lambda$ and $b \neq \lambda$. The internal phase ends and an output symbol is generated. The input symbol is still ignored though.

An internal phase can be seen as a brief time-out for the ITM/A. A user in need of a time-out can pass λ's to the input port, which are ignored by the ITM/A.

Definition 7. *A* time-out *is a maximal part of a computation that receives only λ's on the input port or an internal phase.*

Since we want the machine to be interactive, we require that all time-outs are finite. If an infinite string x is given as input to an ITM/A, then x contains infinitely many symbols not equal to λ, since all time-outs must be finite. The same argument shows that the corresponding output is infinite.

An ITM/A also has an advice mechanism. This mechanism comes in the form of an advice function.

Definition 8. *An* advice function *is a mapping from $\mathbb{N}$ to $\Sigma^\star$.*

The mechanism uses two special tapes and a designated *advice state*. On one of the two tapes, the *advice input tape*, the ITM/A can write the binary representation of an integer m, one bit at a time. At the n-th step, m may not exceed n. At any time during an internal phase, the ITM/A can enter the advice state. Upon entering this state, the string $\alpha(m)$ is written (in one step) on the other tape, the *advice output tape*, replacing any previous contents. In the same step, the contents of the first tape are erased. The contents of the advice output tape can be read normally, but they can only be modified by entering the advice state.

One of the reasons that out of all available models, the ITM/A's have grown so popular, is the fact that their close relation to Turing machines allows us to easily define complexity measures on the ITM/A's. This allows us to look at the efficiency of other models, using the relation to ITM/A as an indirect measure. When it is not so obvious to define a measure of complexity on a model, this indirect approach can yield some insights. We define the following complexity measures on ITM/A's:

Definition 9. *Let $S : \mathbb{N} \to \mathbb{N}$ be a function. An ITM/A is of space complexity S if no more than $S(n)$ tape-cells are ever needed to process n non-empty input symbols. The contents of the advice input tape are taken into account for this complexity, but the contents of the advice output tape are disregarded.*

Definition 10. *Let $f : \mathbb{N} \to \mathbb{N}$ be a function. An ITM/A is of advice complexity f if the length of $\alpha(n)$ is at most $f(n)$ for every n.*

Interactive Turing machines with advice were shown to be equivalent to several other models of evolving interactive systems, see [4, 6, 12]. In this sense, they are an important model. Any potential model of evolving interactive computing can be validated by showing that it realizes the same class of translations as the ITM/A's. We now show that any lineage

[6] Think of it as a filter, which replaces the input symbols by λ before passing them on to the ITM/A. The filter is only active when the ITM/A is in an internal configuration.

of automata can be simulated by an ITM/A, and that any ITM/A can be simulated by a lineage. This result has been sketched in [6], here we refine both the model of the ITM/A and the complexity aspects of the simulation.

Theorem 4. *Let $\Phi : \Sigma^\infty \to \Omega^\infty$ be a translation. Suppose Φ is non-uniform realized by a lineage of automata of complexity g. Then Φ can be realized by an ITM/A of space complexity $O(\log n + \log g)$ and advice complexity $O(g \log g)$.*

Proof. Suppose Φ is non-uniform realized by a lineage $\mathcal{A}$ of complexity g. Let $\alpha : \mathbb{N} \to \Sigma^\star$ be the advice function such that $\alpha(n)$ is the description of A_n. The description is a list of all the states, together with all transitions from each state, see Table 1 for an example. Since

Table 1. Description of A_1 and A_2 from Fig. 1. Each row consists of the name(s) of a state, whether it is an entry or an exit state and a list of transitions from the state. A transition consists of an input symbol, an output symbol and a destination state.

Description of A_1

name	entry	exit	transitions
1	i	-	$(a, b, 2) \to (b, a, 3) \to$ NULL
2	-	-	$(a, a, 4) \to$ NULL
3	-	-	$(b, a, 5) \to$ NULL
4,1	-	o	NULL
5,2	-	o	NULL

Description of A_2

name	entry	exit	transitions
1,1	i	o	$(a, a, 2) \to (b, a, 4) \to$ NULL
2	i	-	$(a, b, 1) \to (b, a, 3) \to$ NULL
3	-	-	$(a, a, 1) \to (b, b, 4) \to$ NULL
4,2	-	o	NULL

there are $g(n)$ states, we can describe each state with $O(\log g(n))$ bits. Each state requires a constant extra amount of bits to indicate if it is an entry or exit state. To encode a transition, we need the in- and output symbols, and the destination state. Thus we need $O(\log g(n))$ bits per transition. Since Σ is finite, there can at most be a constant number of transition from any one state. It follows that $O(g(n) \log g(n))$ bits are enough to describe the automaton.

The labels of the exit states of A_n need to correspond to the names of the entry states of A_{n+1}. Thus, we label the exit states with two names, one name to be used in A_n and one for use in A_{n+1}. By using a variable-length encoding (see [7] for details) and choosing the names for the exit states wisely, we can ensure that the size of the two names together is $O(\log g(n))$.

We construct an ITM/A M that uses the advice value $\alpha(n)$ to simulate the automaton A_n. The simulation is mostly done in internal phases. Only the actual reading and writing of symbols is done externally.

First, M determines the value of A_1. This is done internally by writing the index 1 to the advice input tape and entering the advice state. Then the description of A_1 appears on the advice output tape. The simulation starts by looking up the initial state of A_1, and writing it to a work tape. Then M reads the first input symbol. Now, using the advice, M can find the next state and the corresponding output symbol, in another internal phase. Then M looks up this state, and writes it to the work tape, overwriting the previous contents. Then M writes the output symbol to the output port. Now, M checks if the new state is an exit state. If this is not the case, then M repeats the process. Otherwise, M increases the index by one on the advice input tape and enters the advice state, to obtain the description of A_2. Then M looks up the state in this update, which exists because $\mathcal{A}$ is a lineage. Now M can repeat the process with this update.

It is left to the reader to verify that the translation realized by M matches the translation from $\mathcal{A}$. We see that the advice function has size $O(g \log g)$, and the work space uses $O(\log n + \log g)$ cells. Moving from state to state, and updating the automaton can all be done in finite time, so the internal phases are finite, and M is a valid ITM/A.

□

Theorem 5. *Let $\Phi : \Sigma^\omega \to \Omega^\omega$ be a translation. Suppose Φ is realized by an ITM/A with k tapes, with a space complexity $g(n)$ and advice complexity $f(n)$. Then Φ can be non-uniform realized by a lineage of automata of complexity*

$$O\left(c^{kg(n)} g(n)^k f(n) n^2 \log(n)\right) \quad , \tag{26}$$

where c is the size of Σ.

Proof. Suppose Φ is realized by an ITM/A M. We will simulate M with a lineage $\mathcal{A}$. Every state of A_n will correspond to an external configuration of M, with the contents of the advice tapes included. Each state is an exit state.

On a prefix of length n, M can ask for advice only for the values 1 through n. Hence the advice tapes have at most n different contents, and the heads can be in one of $f(n)$ or $\log n$ positions, respectively. Since the work space of M is bounded by $g(n)$, each of the remaining tapes has at most $c^{g(n)}$ different contents and the head can be in one of the $g(n)$ positions. It follows that there are

$$O\left(g(n)^k f(n) \log(n) \cdot c^{kg(n)} n^2\right)$$

possible configurations. Now the simulation takes place by moving from configuration to configuration, following the transition function from M.

The problem with this approach is that internal phases cannot be simulated in real-time, because there are no states corresponding to internal configurations. This is intentional, because either the input or the output is non-existent in a transition involving such a configuration. But since an internal phase is finite and the ITM/A is deterministic, the whole internal phase, which in fact consists of an external configuration s followed by a finite sequence of internal configurations and which is concluded by an external configuration r, can be simulated by one transition from s to r. Conveniently, during an internal phase, only one non-empty input and output symbol appear. Naturally, these symbols accompany the transition from s to r in the simulation.

□

When we compare the results of Theorems 4 and 5, we see that an ITM/A can be more efficient than a lineage of automata. Applications of the Theorems typically involve classes of functions. For example, a lineage of polynomial complexity can be simulated by an ITM/A with logarithmic space and polynomially bounded advice, and vice versa. Similarly, a lineage of exponential complexity can be simulated by an ITM/A with polynomial space and exponentially bounded advice. Using Prop. 9, an ITM/A of those complexities can also be simulated by a lineage of exponential complexity.

7 Open Problems

In this paper, we didn't impose any restrictions on the structure of the automata, or the amount of change (other than a sheer size restriction) that was permitted in one update. While this gives us a tremendous freedom, it is not realistic. Many examples of evolving interactive systems are highly structured. Much study has already gone into the structure of the Internet. It is unknown what effects these structures have on our model. Results similar to that of section 5 for the ITM/A are still open problems.

8 Conclusion

In this paper, we have developed the theory of lineages as a new model of computation. A lineage is a sequence of finite automata, where each automaton in the sequence is viewed as the next instantiation or incarnation of the evolving system that it models. By using lineages, one can immediately single out the evolutionary aspects of the system. The development of a lineage is modelled by looking at the automata in the sequence, and the relation between each automaton and its immediate successor.

Lineages are a model for evolving interactive systems. We have established their position among the other models of evolving interactive systems by showing their equivalence to interactive Turing machines with advice. We have shown that lineages, like all models of evolving interactive systems, are more powerful than traditional finite-state models and gave several operations on lineages.

An important characteristic of an automaton is its size. For a lineage, we defined a complexity measure based on the size of the automata in the sequence. We proved in Thm. 3 that lineages of higher complexity are able to l-realize more translations than lineages of lower complexity. Specifically, for each non-decreasing function g, there is a translation that can be l-realized by a lineage of complexity g, but not by any lineage that has fewer than $g(n)$ states available for its n-th automaton, for a certain n. On the other hand, once a translation (over an input alphabet of size c) is fixed, we know that it can be l-realized by a lineage of complexity at most c^n.

The equivalence between lineages and interactive Turing machines with advice allows us to compare the efficiency of the two models, expressed by the complexity measure. It turns out that ITM/A's are logarithmically more efficient than lineages in space-usage, while the length of the advice corresponds to the size of the automata of the lineage.

We conclude that, while there are more efficient models for evolving interactive systems, lineages of automata are rather more attractive than other models, since the important aspects of the system are easily found in the model. Furthermore, finite automata are well-studied, and many results can be adapted to the theory of lineages.

References

1. J.L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I*, 2nd Edition, Springer-Verlag, Berlin, 1995.

2. L.H. Landweber. Decision Problems for ω-Automata, *Math. Syst. Theory*, Vol. 3 No. 4, Springer-Verlag, New York, 1969, pp. 376-384.
3. J. van Leeuwen, J. Wiedermann. On Algorithms and Interaction, in: M. Nielsen, B. Rovan (Eds.), *Mathematical Foundations of Computer Science 2000*, 25th Int. Symposium, Lecture Notes in Computer Science Vol. 1893, Springer-Verlag, Berlin, 2000, pp. 99-113.
4. J. van Leeuwen, J. Wiedermann. The Turing Machine Paradigm in Contemporary Computing, in: B. Enquist, W. Schmidt (Eds.), *Mathematics Unlimited - 2001 and Beyond*, Springer-Verlag, Berlin, 2001, pp. 1139-1156.
5. J. van Leeuwen, J, Wiedermann. A Computational Model of Interaction in Embedded Systems, in: Technical Report UU-CS-2001-02, Institute of Information and Computing Sciences, Utrecht University, 2001.
6. J. van Leeuwen, J. Wiedermann. Beyond the Turing Limit: Evolving Interactive Systems, in: L. Pacholski, P. Ružička (Eds.), *SOFSEM 2001: Theory and Practice of Informatics*, 28th Conference on Current Trends in Theory and Practice of Informatics, Lecture Notes in Computer Science Vol. 2234, Springer-Verlag, Berlin, 2001, pp. 90-109.
7. M. Li, P. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*, 2nd Edition, Springer-Verlag, New York, 1997.
8. L. Staiger. ω-Languages, in: G. Rozenberg, A. Salomaa (Eds.), *Handbook of Formal Languages*, Vol. 3, *Beyond Words*, Springer-Verlag, Berlin, 1997, pp. 339-387.
9. W. Thomas. Automata on Infinite Objects, in: J. van Leeuwen (Ed.), *Handbook of Theoretical Computer Science*, Vol. B, Elsevier Science, Amsterdam, 1990, pp. 134-191.
10. P. Wegner, D. Goldin. Computations beyond Turing Machines, *Communications of the ACM* Vol. 46, Springer-Verlag, New York, 2003, pp. 100-102.
11. J. Wiedermann, J. van Leeuwen. Emergence of a Super-Turing Computational Potential in Artificial Living Systems, in: J. Kelemen, P. Sosík (Eds.), *Advances in Artificial Life*, 6th European Conference (ECAL 2001), Lecture Notes in Artificial Intelligence Vol. 2159, Springer-Verlag, Berlin, 2001, pp. 55-65.
12. J. Wiedermann, J. van Leeuwen. The Emergent Computational Potential of Evolving Artificial Living Systems, *AI Communications*, Vol. 15 No. 4, IOS Press, Amsterdam, 2002, pp. 205-215.