The EMERGENT HYPERCOMPUTATIONAL POWER of ARTIFICIAL LIVING SYSTEMS

Master's Thesis
ICA-3175464
Technical Artificial Intelligence
Department of Computer Science, Utrecht University

Author:
Susanne van den Elsen

Supervisor:
Dr. Gerard Vreeswijk

Second reviewer:
Dr. Mehdi Dastani
Third reviewer:
Prof. Dr. Jan van Leeuwen

2012

Acknowledgements

A word of gratitude goes to those people without whom this thesis would not have come about in its current shape. First of all, I would like to thank my supervisor Gerard Vreeswijk for guiding me throughout the process, from the day I set my first steps into the world of hypercomputation, to the day I killed my last dear darling, for sharing his time and insights in valuable discussions. I would also like to express my gratitude to Jan van Leeuwen and Jiří Wiedermann for having the opportunity to discuss their work with them. Thanks go to Clemens Grabmayer, for being willing to discuss some of the puzzling problems I encountered in several stages of my project, as well as for giving me the opportunity to present my findings in a guest lecture in the master's course "Models of computation".

I want to thank Yannick Bitane for luring me to the university library on a daily basis, with a promise of coffee and refreshing conversations. Lucy Hagen and Annabelle van den Elsen for always knowing the right things to say, for their moral support. My parents, for supporting me in various ways during my years of education. Last, but not least, and right from the heart, I want to say "thank you ♡" to Hilke Haase for sharing my tears, as well as my laughs.

Contents

List of figures and tables

Figures

All figures in this paper are original and created by the author.

Tables

Chapter 1
Introduction

Computability theory, which is concerned with the question what is and what is not computable, has since the beginning of the twentieth century been based on the Turing machine [56] model. The universal Turing machine, a machine which can simulate all other Turing machines, is the conceptual basis of general purpose computers. Turing's thesis [56] states that everything which is effectively computable, or alternatively, computable in an algorithmic or mechanistic way, can be computed by some Turing machine. This has led to a common belief in computer science that the Turing machine is a model for *all* computations, and for computations performed by computers in specific. However, the notion of computation has changed dramatically over the past decades. On the one hand, contemporary computing systems are being inspired more and more by nature. System such as large computer networks, and multi-agent systems are massively parallel, open systems, composed of interacting components. On the other hand, natural living systems are also frequently being regarded as information processing, or computing systems. The change in the nature of computing systems, as well as the computations they perform, leads some to argue that the Turing machine is no longer a suitable model for contemporary computing systems. Van Leeuwen and Wiedermann argue that "the classical Turing paradigm may no longer be fully appropriate to capture all features of present-day computing" [61, p. 5]. Likewise, Goldin and Wegner argue that "[i]t is time to recognize that today's computing applications, such as web services, intelligent agents, operating systems, and graphical user interfaces, cannot be modelled by Turing machines; alternative models are needed" [24, p. 153]. With alternative models may come alternative notions of what is computable. The field of hypercomputation is concerned with studying models which compute more than Turing machines, or are incomparable to Turing machines.

In 2001 and 2002, Van Leeuwen and Wiedermann [68, 69] claimed that hypercomputational power can emerge in artificial living systems. In their articles, Van Leeuwen and Wiedermann present computational models of individual living organisms — active cognitive transducers— and "consider AL systems composed of evolving communities of communicating active cognitive transducers" [69, p. 206]. The models compute under "a certain non-commonly considered computational scenario" [68, p. 56], called "*evolving interactive computing*" [62, p. 91], which combines the three features of interactivity, non-uniform evolution, and infinity of operation. Then, Van Leeuwen and Wiedermann "show that a 'super-Turing' computational power can indeed emerge in such systems" [69, p. 206]. Van Leeuwen and Wiedermann argue that their artificial living systems have hypercomputational power by showing that they are equally powerful as interactive Turing machines with advice [61, 62]. Interactive Turing machines with advice compute with information from a special kind of oracle, which makes them more powerful than standard interactive Turing machines. According to Van Leeuwen and Wiedermann, the hypercomputational power of their artificial living systems is emergent. In the authors' words, "a community of active cognitive transducers has a much greater computing power than just the 'sum' of the powers of the individual transducers. Here we see the emerging super-recursive computing power" [69, p. 213]. One of Van Leeuwen and Wiedermann's conclusions is that "[t]he information processing capabilities of AL systems are far more powerful than commonly believed" [69, p. 213].

The three features of evolving interactive computing — interactivity, non-uniform evolution, and infinity of operation — have been studied previously in isolation. The feature of infinite operation has been studied, among others, by Thomas [55] in the form of a theory of ω-automata. A paradigm shift from rule-based algorithms towards interactive computation was argued for by Wegner in his seminal paper [66], and by Goldin and Wegner [24]. Wegner [66] claims that interaction machines are more powerful than standard Turing machines. A theory of interaction was initiated by Van Leeuwen and Wiedermann [60] and further developed in [63]. Van Leeuwen and Wiedermann [60] suggest that interaction alone does not lead to super-Turing computing power, but rather "leads to a generalization of standard computability theory to the case of infinite computations" [68, p. 63]. The aspect of non-uniform evolution draws from non-uniform complexity theory, which has been studied, among others, by Karp and Lipton [31]. The combination of interaction, non-uniform evolution, and infinity of operation, and the evolving interactive paradigm, was previously considered by Van Leeuwen and Wiedermann [61, 62] in a different context. The complexity of evolving systems is studied by Verbaan [64].

Van Leeuwen and Wiedermann's claim that hypercomputational power can emerge in artificial living systems involves three notions: hypercomputation, artificial life, and emergence. Each of these three notions is studied extensively independently of the other notions. The term 'hypercomputation' is introduced by Copeland and Proudfoot in [18]. Examples of theoretical models claimed to possess hypercomputational power, and related to Van Leeuwen and Wiedermann's models, include Turing's [57] o-machines, although the claim was by Copeland [16], infinite time Turing machines by Hamkins and Lewis [25], and asynchronous networks of Turing machines by Copeland and Sylvan [19]. An assessment of the resources used by a selection of hypercomputational models from the literature, their physical realisability, exploitability and relative power is made by Ord [42]. Hypercomputation is a controversial subject, which has passionate proponents, as well as opponents. The physical realisability and exploitability of hypercomputation are arguably its most controversial aspects. Stannett [53], among others, argues that hypercomputation is not in principle impossible, and that irrespective of its physical possibility, it is a more insightful model of mathematical, physical, and biological processes than classical computation. For Davis [22], on the other hand, the implausibility of exploitable hypercomputation is reason to argue that there is no such subject as hypercomputation.

Artificial living systems are studied in the field of ALife, which encompasses a very broad and diverse range of research efforts. Artificial living systems and research efforts in the literature include cellular automata, originally due to Von Neumann [65], genetic algorithms (see for example Holland [28]), and biologically inspired robots (see for example Brooks [8]). However, this list is far from an exhaustive one. An extensive study of cellular automaton behaviour, and the relation with computability theory is carried out by Wolfram [72], Langton [35] and Cook [11], among others.

The third and final component of Van Leeuwen and Wiedermann's claim is emergence. Emergence has first been comprehensively worked out by a number of British scientists, nowadays often called 'British emergentists' [41]. These British emergentists include Mill [38], Broad [7], Morgan [40], and Alexander [1]. In recent years, the term 'emergence' is being used in a variety of contexts, while according to Cooper, it has "generated more questions than answers, and more excitement than clarity" [13, p. 194]. Humphreys [30] has made a twofold taxonomy of existing approaches to emergence: the relational taxonomy and the temporal taxonomy. Bedau [3] has formulated a notion of weak emergence, which is especially relevant to ALife.

Although the three notions involved in Van Leeuwen and Wiedermann's claim — hypercomputation, artificial life, and emergence — are widely studied, they do not have generally accepted definitions or theories. The main goal of this thesis is to review Van Leeuwen and Wiedermann's claim against a more detailed discussion of each of its components. The aim of this review is to see whether Van Leeuwen and Wiedermann's claim is indeed tenable. More specifically, we first discuss whether Van Leeuwen and Wiedermann's models can indeed be considered artificial living systems. Second, we discuss whether Van Leeuwen and Wiedermann's models are indeed capable of hypercomputation. Third, we examine whether Van Leeuwen and Wiedermann's results are indeed a case of emergence. Although Van Leeuwen and Wiedermann's approach is a computability-theoretic one, their claim lies on the intersection between computability theory, ALife, philosophy, and physics. In our discussion, we therefore take an interdisciplinary approach.

The outline of this thesis is as follows. First, we give an exposition of Van Leeuwen and Wiedermann's argument for their claim in Chapter 2. In Chapter 3, we discuss the notion of hypercomputation. In Chapter 4, we discuss artificial living systems and the field of ALife. In Chapter 5, we discuss a number of notions of emergence. In Chapter 6, we discuss Van Leeuwen and Wiedermann's claim against the background of our discussion of hypercomputation, ALife, and emergence.

Chapter 2
Emergent hypercomputational power in AL systems

In this chapter, we give an exposition of Van Leeuwen and Wiedermann's [68, 69] formalisation of their claim that hypercomputational power can emerge in artificial living systems. First, we introduce two models of artificial living systems presented by the authors: lineages of cognitive transducers and communities of active cognitive transducers. Second, we introduce the interactive Turing machine with advice, which Van Leeuwen and Wiedermann use for benchmarking the computational power of their models of artificial living systems. Finally, we show how Van Leeuwen and Wiedermann prove that the computational power of lineages of cognitive transducers, and communities of active cognitive transducers is equal to that of interactive Turing machines with advice, which shows that they possess hypercomputational power.

2.1 Models of artificial living systems

In this section, we introduce two models of artificial living systems presented by Van Leeuwen and Wiedermann [68, 69]: lineages of cognitive transducers, which model evolution of organisms, and communities of active cognitive transducers, which model conglomerates of organisms. First, we introduce Van Leeuwen and Wiedermann's model of an individual organism: the cognitive transducer, which is the building block for lineages and communities.

Cognitive transducers

In [68, 69], Van Leeuwen and Wiedermann present their model of an individual living organism: the cognitive transducer. The authors seem to be aware that, when modelling a physical information processing system in order to investigate its computational power, it is important that the model neither overestimates, nor underestimates the computational power of the system at hand. Basically, a cognitive transducer is a finite discrete-state machine. The authors' choice for finite discrete-state machines is based on their assumption that living organisms can only enter into a finite number of different internal configurations. Since living organisms both receive input and produce output, cognitive transducers are finite state machines extended with the ability to produce output based on the input and their internal state.

Van Leeuwen and Wiedermann point out that the computational scenario of cognitive transducers has to reflect the scenario under which living organisms process information. This scenario, they argue, is fundamentally different from the computational scenario of standard finite state machines.

Van Leeuwen and Wiedermann point out that standard finite state machines perform their computation on a finite input string, which is fixed on their input tape prior to the start of the computation. This input string is not allowed to be changed during computation, not even the not-yet read parts of it. Moreover, the computation of standard finite state machines terminates after a finite number of steps. The machines are not able to transfer information they computed from one computation to the next. Every time a new computation is started, the machine starts from the same initial state. This prevents standard finite state machines from learning, because given the same input, the machine will always compute the same output.

Living organisms, on the other hand, receive their inputs from their environment constantly, and in an on-line manner, via their sensory systems. These sensory systems can consist of multiple channels, depending on the organism's complexity. Inputs thus 'stream' into organisms in a parallel way, and the organism also processes them in a parallel manner. The computation performed by organisms does not terminate in principle. The computation only terminates when the organism dies. An organism's output does not only depend on the current input, but on the history of inputs up to now. Since organisms can modify their environment or communicate with other systems in their environment, the inputs they receive may depend on their own previous outputs, as well as the reactions of other organisms. This information processing scenario, which the authors [68, p. 58] call 'perpetual interactive learning', Van Leeuwen and Wiedermann state, leads to an organism's potential to learn from experience.

Taking this information processing scenario into account, Van Leeuwen and Wiedermann present the interactive finite state transducer as paradigmatic example of a cognitive transducer. The interactive finite state transducer is a generalisation of a Mealy machine, equipped with input and output ports rather than tapes and tape heads, which allow the transducer to interact with its environment. Symbols arrive one at a time at the input port, and are produced by the transducer at the output port in a similar way. Based on this description, we define interactive finite state transducers as follows.

Definition 2.1.1 (Interactive finite state transducer). *An interactive finite state transducer (IFT) with k input ports and l output ports is a quintuple $\langle Q, \Sigma, \Gamma, q_0, \delta\rangle$, where*

- *Q is a finite set of states;*
- *Σ is a non-empty finite input alphabet;*
- *Γ is a non-empty finite output alphabet;*
- *$q_0 \in Q$ is the initial state;*
- *$\delta : Q \times (\Sigma \cup \{\lambda\})^k \to Q \times (\Gamma \cup \{\lambda\})^l$ is a partial transition function.*

Throughout their papers, Van Leeuwen and Wiedermann interpret λ as a fixed symbol meaning, when appearing at an input or output port, that there is actually no meaningful symbol at the respective port.

Van Leeuwen and Wiedermann note that, instead of Mealy machines, other finite-state devices, such as discrete neural networks, can lay at the basis of a cognitive transducer. This motivates the following definition of the class of cognitive transducers [69, p. 208].

Definition 2.1.2 (Cognitive transducer)**.** *The class $\mathcal{CT}$ of cognitive transducers is inductively defined as the minimal class that satisfies the following rules:*

1. *interactive finite transducers (IFTs) are in $\mathcal{CT}$, and*
2. *any device computationally equivalent to IFTs is in $\mathcal{CT}$.*

Throughout our discussion, we use the interactive finite state transducer as the paradigmatic example of cognitive transducers. Based on [60], we describe an interactive computation of a cognitive transducer and its environment as a translation of input streams from the environment to output streams, which are the cognitive transducer's response to the input. Since cognitive transducers operate infinitely, at least in principle, they translate infinite input strings into infinite output strings. In what follows, we refer to such infinite strings as 'streams', denote the set of streams over alphabet Σ by Σ^ω, and denote a stream $x \in \Sigma^\omega$ by $x_1x_2\ldots$. Hence, a cognitive transducer realises a partial translation $\Phi_{k,l} : ((\Sigma \cup \{\lambda\})^k)^\omega \to ((\Gamma \cup \{\lambda\})^l)^\omega$. Since cognitive transducers are basically finite state machines, the translations realisable by the class of cognitive transducers are called 'regular translations' [68].

Definition 2.1.3 (Regular translation)**.** *A regular translation is a translation $\Phi_{k,l} : ((\Sigma \cup \{\lambda\})^k)^\omega \to ((\Gamma \cup \{\lambda\})^l)^\omega$ realised by a cognitive transducer.*

Without loss of generality, we assume that $k = l = 1$ in what follows.

Because we need it later on, we discuss here how each cognitive transducer can be encoded into a finite string in an effective way.

Definition 2.1.4 (Encoding of a cognitive transducer)**.** *Let $M = \langle Q, \Sigma, \Gamma, q_0, \delta\rangle$ be a cognitive transducer, and let $|Q| = m, |\Sigma| = k$, and $|\Gamma| = l$. Then an encoding $\langle M\rangle$ of M is a binary string containing*

- *a list of states, with each $q_i \in Q$ represented by the number i in binary. The total list is described by a string of length $m \log m$;*
- *a list of tuples $\langle q, \sigma, q', \gamma\rangle \in \delta$, with states $q, q' \in Q$, symbol $\sigma \in \Sigma$, and symbol $\gamma \in \Gamma$. There are $l(m + 1)^{km}$ possible transition tuples, each described by a string of length $2 \log m + \log k + \log l$;*
- *a list containing the initial state, represented by a string of length $\log m$.*

Active cognitive transducers

Van Leeuwen and Wiedermann extend cognitive transducers to incorporate the ability of living organisms to "influence and *modify* the environment in which they operate" [69, p. 211]. Thereto, the authors provide cognitive transducers with the ability to move around in their potentially infinite environment, for example a two-way infinite tape, and mark cells in that environment with symbols from a finite marking alphabet. Placed marks can be read from the environment at a later time by the transducer. This extension yields active cognitive transducers, also called 'agents' by the authors.

Van Leeuwen and Wiedermann [69, p. 211] explain that "[m]odels of finite transducers with a two-way input tape which can mark cells on their input tape have been studied for years in automata theory" and that "[t]he machines are computationally provably more powerful than their non-marking counterparts". However, they do not explain how these marking finite transducers are extended to their interactive versions. Still, Van Leeuwen and Wiedermann [69, p. 211] state that "[a]ctive cognitive transducers have a computational power equivalent to interactive Turing machines". We introduce interactive Turing machines in Section 2.2.

We give the following formal definition of active cognitive transducers.

Definition 2.1.5 (Active cognitive transducer). *An active cognitive transducer is a* 6*-tuple* $\langle Q, \Sigma, \Gamma, \Pi, q_0, \delta \rangle$*, where*

- Q *is finite set of states;*
- Σ *is a non-empty finite input alphabet;*
- Γ *is a non-empty finite output alphabet;*
- Π *is a non-empty finite marking alphabet;*
- $q_0 \in Q$ *is the initial state;*
- $\delta : Q \times (\Pi \cup \{\varepsilon\}) \times (\Sigma \cup \{\lambda\}) \to Q \times (\Pi \cup \{\varepsilon\}) \times \{L, R, none\}^k \times (\Gamma \cup \{\lambda\})$ *is a partial transition function, where* ε *is the empty symbol, and* $k \in \mathbb{N}$ *the dimension of movement.*

If the active cognitive transducer is non-deterministic, δ is a transition relation rather than a function, and $\delta \subseteq Q \times (\Pi \cup \{\varepsilon\}) \times (\Sigma \cup \{\lambda\}) \times Q \times (\Pi \cup \{\varepsilon\}) \times \{L, R, none\}^k \times (\Gamma \cup \{\lambda\})$.

Although an active cognitive transducer is not equipped with a work tape, like for example a Turing machine, it can use its environment as a kind of work tape. Rather than moving a tape head over the work tape, the active cognitive transducer moves around in the environment itself. It is thus necessary to define an environment.

Definition 2.1.6 (Grid environment). *A grid environment is a tuple* $\langle \Pi, \mathcal{L} \rangle$*, where*

- Π *is a non-empty finite environment alphabet;*
- $\mathcal{L} = \mathbb{Z}^k$ *is a k-dimensional grid of cells, each containing a symbol of* $\Pi \cup \{\varepsilon\}$*, where* ε *denotes the empty symbol.*

Now, we can define a configuration of an active cognitive transducer.

Definition 2.1.7 (Active cognitive transducer configuration). *Let* $M = \langle Q, \Sigma, \Gamma, \Pi, q_0, \delta \rangle$ *be an active cognitive transducer, and* $E = \{\Pi, \mathcal{L}\}$ *its k-dimensional grid environment. A configuration of* M *is a tuple* $c = \langle q, p \rangle$*, where*

- $q \in Q$ *is the current state of* M*;*
- $p = \langle p_1, ..., p_k \rangle \in \mathbb{Z}^k$ *is the current location of* M *in* E*.*

Remark 2.1.1. There seem to be two different notions of 'environment' in Van Leeuwen and Wiedermann's [69] article. The first notion refers to the environment with which an (active) cognitive transducer interacts: the interaction partner. To this notion, the environment seems to be an information processing entity itself, which "can behave like an adversary and send signals in any arbitrary, unpredictable and 'non-algorithmic' way" [60, p. 102]. The second notion of environment seems to refer to an external memory, possibly with private as well as public parts. This notion of environment is modelled by our grid environment. One could think of an active cognitive transducer sharing the public parts of a grid environment with its interaction partners. In this way, the grid environment can facilitate indirect communication between the active cognitive transducer and its interaction partners.

Lineages

With a model of individual living organisms in place, Van Leeuwen and Wiedermann [69] use lineages of cognitive transducers as a means to model the evolution of living organisms. Lineages of interactive finite automata were introduced by Van Leeuwen and Wiedermann in an earlier article [62] in the context of presenting a new model of computation that captures the features of present-day computing. The following definition is from [69, p. 209]. Let $\mathcal{U}$ be some universe of possible states.

Definition 2.1.8 (Lineage of cognitive transducers)**.** *Let $\mathcal{A} = \{A_1, A_2, ...\}$ be a lineage of IFTs [cognitive transducers] over Σ, and let $Q_i \subseteq \mathcal{U}$ be the set of states of A_i. Let $G = \{G_1, G_2, ...\}$ be a lineage of non-empty finite sets of $\mathcal{U}$ such that $G_i \subset Q_i$ and $G_i \subseteq G_{i+1}$, for $i \geq 1$. Then $\mathcal{A}$ with G is called a lineage of IFTs [cognitive transducers].*

In what follows, we will often omit G from explicit mention. Moreover, we will often abbreviate 'lineages of cognitive transducers' and just speak of 'lineages'.

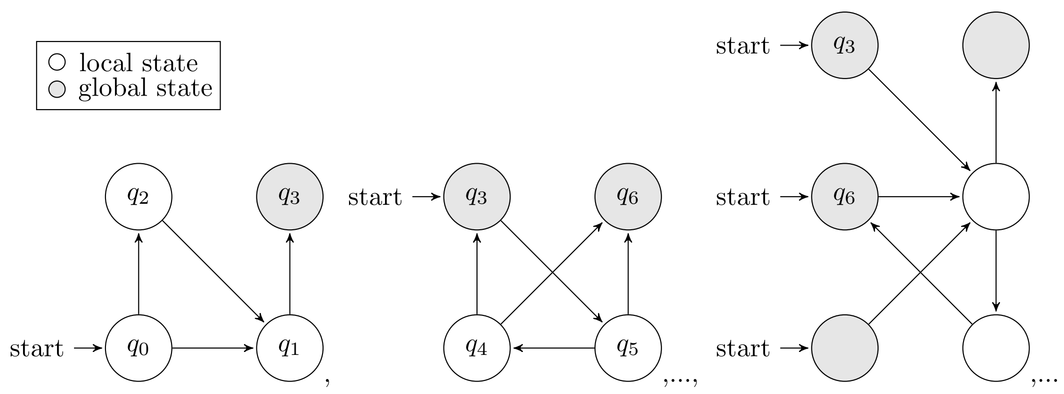


Figure 2.1: Illustration of a lineage $\mathcal{A} = \{A_1, A_2, ..., A_n, ...\}$ with global states $G = \{q_3, q_6, ...\}$.

Given an input stream $x \in (\Sigma \cup \{\lambda\})^\omega$, a lineage $\mathcal{A}$ computes as follows. Initially, A_1 is the controlling cognitive transducer. In general, if A_i is the controlling cognitive transducer it performs computation using its local states, i.e. $Q_i - G_i$, until it reaches a global state

$g \in G_i$. It then passes on control to cognitive transducer A_{i+1}, who starts in state $g \in G_{i+1}$, and continues reading the next symbol of the input stream.

The property of evolution is modelled by the fact that the next cognitive transducer in the lineage has a possibly richer set of internal configurations than the previous one. With passing over the control from one 'generation' i to the next, information assembled by the previous generations is preserved, since A_{i+1} starts in the same state as M_i stopped.

Instead of considering lineages of cognitive transducers, and model evolution in terms of generations of automata, Van Leeuwen and Wiedermann note that one can also consider the evolution of individual cognitive transducers. The cognitive transducer is then defined so that it simulates A_i if and only if $A_i \in \mathcal{A}$ is the currently controlling cognitive transducer. In parallel with lineages of cognitive transducers, in general, A_i remains active until it reaches a global state $g \in G_i$. Then, A_{i+1} becomes the controlling machine, and the cognitive transducer proceeds by simulating A_{i+1} from the state it is currently in.

It is important to note that Van Leeuwen and Wiedermann assume that there need not to be an algorithmic way to compute the description of the cognitive transducer A_i in a lineage given index i, or given the description of the previous elements in the lineage. Enumerating all members of the lineage may thus be the only way to describe the lineage. However, Van Leeuwen [58] adds that this is a worst case scenario. He also points to the possibility to require the lineage to be bounded, for example by a polynomial function, to make it a more plausible model for evolution. This goes beyond the scope of our thesis. For an account on lineages of general classes of machines, as well as complexity classes for lineages of machines, we refer to [64].

Van Leeuwen and Wiedermann point out that, although lineages are infinite objects, at each time t, only one member of a lineage — a finite object — is actively engaged in the computation.

Communities

Having introduced cognitive transducers and active cognitive transducers as a model for individual living organisms, and lineages as a means to model their evolution or adaptation, Van Leeuwen and Wiedermann [68, p. 61] state that "[u]ltimately, active cognitive transducers are of interest only in large conglomerates, interacting like 'agents' of individually limited powers." Finally, the authors present their model for artificial living systems: communities of active cognitive transducers.

Van Leeuwen and Wiedermann [68, 69] define a community of active cognitive transducers as a set of active cognitive transducers, which varies over time, but is at each time finite. The active cognitive transducers in the community share the same environment, in which they can move around, as well as write and read symbols from a finite alphabet. In this way, the active cognitive transducers in the community can communicate with each other. Van Leeuwen and Wiedermann suggest that other mechanisms for communication, such as an Internet-like infrastructure, mobile phones, or snail-mail, can be incorporated into the model as well. Depending on the chosen mechanism, the active cognitive transducers need to be extended, for example with a special message port, to facilitate communication using the mechanism. However, the authors do not explicitly incorporate a specific mechanism for communication in their model. In what follows, we assume the agents communicate by writing messages in the environment, such that they can be read by other agents. Van Leeuwen and Wiedermann assume that communication between active cognitive transducers in the community may proceed unpredictably. The authors write that "[o]ne can see it also as if the agents move in their environment and encounter each other randomly, unpredictably, or intentionally, and exchange messages" [68, p. 61]. Van Leeuwen [58] points out that this unpredictability can be the result of for example asynchrony in the community, assuming that each agent has its own internal clock, or non-determinism in the agents' transition function.

Based on the informal description in [68, 69], we formalise the definition of communities of active cognitive transducers as follows.

Definition 2.1.9 (Community of active cognitive transducers)**.** *A community of active cognitive transducers is a triple* $\mathcal{G} = \langle \beta, \Delta, E \rangle$*, where*

- $\beta : \mathbb{N} \to 2^{\mathbb{N} \times \Sigma^*}$ *is the description function. The description function assigns to each time* $t \geq 0$ *the finite set* $\beta(t) = \{\langle i, \langle M_i \rangle \rangle \mid$ *active cognitive transducer* M_i *with id* i *and encoding* $\langle M_i \rangle$ *is in the community at time* $t\}$*. Thus, at each time* t*, community* $\mathcal{G}$ *has* $|\beta(t)|$ *members, i.e. the active cognitive transducers in the set* $\{M_i \mid \langle i, \langle M_i \rangle \rangle \in \beta(t)\}$*;*
- $\Delta : \mathbb{N} \to 2^{\mathbb{N} \times \Sigma^*}$ *is the timing function. The timing function assigns to each time* $t \geq 0$ *a finite subset of* $\beta(t)$*, consisting of the active cognitive transducers that make a transition at time* t*;*
- E *is a grid environment.*

Remark 2.1.2. Note the emphasis on asynchrony in our definition of a community. This emphasis is absent in Van Leeuwen and Wiedermann's [68, 69] informal description. We define asynchronous communities of active cognitive transducers to take the unpredictability of agent communication into account. Note that synchronous communities are a special instance of asynchronous communities, since the former are characterised by a timing function such that $\forall t \in \mathbb{N}(\Delta(t) = \beta(t))$.

Next, we define the description and the configuration of a community of active cognitive transducers.

Definition 2.1.10 (Description of a community of active cognitive transducers)**.** *Let* $\mathcal{G} = \langle \beta, \Delta, E \rangle$ *be a community of active cognitive transducers. The description of the* $\mathcal{G}$ *at time* t *is given by* $\beta(t)$*.*

Definition 2.1.11 (Configuration of a community of active cognitive transducers)**.** *Let* $\mathcal{G} = \langle \beta, \Delta, E \rangle$ *be a community of active cognitive transducers. A configuration of* $\mathcal{G}$ *is a tuple* $\mathbf{c} = \langle C, X \rangle$*, where*

- C *is the set of configurations of the active cognitive transducers currently in the community*
- X *is a* k*-dimensional array of symbols, corresponding to the current contents of the* k*-dimensional grid environment* E*.*

According to Van Leeuwen and Wiedermann's description, the input and output ports of all active cognitive transducers in the community together represent the input and output ports of the community as a whole. Since the size of the community varies over time, so does the number of input and output ports of the community. A community of active cognitive transducers therefore realises a translation over streams of symbol 'packages' of varying size. The input stream is denoted by

$$x = \{(x_{i_1,t}, x_{i_2,t}, ..., x_{i_k,t})\}_{t=0}^{\infty}, \text{ with}$$

$$\{\langle i_1, \langle M_{i_1}\rangle\rangle, \langle i_2, \langle M_{i_2}\rangle\rangle, ..., \langle i_k, \langle M_{i_k}\rangle\rangle\} = \beta(t).$$

The same holds for the community's output stream, which is denoted by

$$y = \{(y_{i_1,t}, y_{i_2,t}, ..., y_{i_k,t})\}_{t=0}^{\infty}.$$

It is important to note that Van Leeuwen and Wiedermann do not require β to be Turing-computable. In fact, they assume that there need not be an effective way to compute the set of id's and encodings of the members of a community — the description of a community — at a given time t. If the description function is indeed not Turing-computable, the only way to have access to the description of the community at time t is to let the community 'evolve' over time, up to time t, and observe the community. On the other hand, this is a worst case scenario. It could still be that β is indeed Turing-computable. The same holds for the timing function Δ.

Moreover, Van Leeuwen and Wiedermann do not require the function β to be bounded in some way. Therefore, there is no restriction on the growth of the size of the community with the processing time.

2.2 Interactive Turing machines with advice

In order to investigate the computational power of lineages of cognitive transducers and communities of active cognitive transducers, they need to be compared to another model, one of which the computational power is known. Van Leeuwen and Wiedermann [68, 69] use the interactive Turing machine with advice as their model for benchmarking. In this section, we introduce the interactive Turing machine with advice.

Interactive Turing machines

First, we discuss the interactive Turing machine. The computational power of interaction machines was studied by Van Leeuwen and Wiedermann in [60]. This study was motivated by Wegner [66, p. 83], who suggested the extension of Turing machines "by addition of input and output actions that support dynamic interaction with an external environment".

An interactive Turing machine has an input port, an output port and k work tapes. The machine is designed in such a way, that at each time t it requires a symbol at its input port. Analogously, at each time the machine produces a symbol at its output port. In the meanwhile, the machine can perform internal computation on the input it has received up to then. Thereto, the machine enters an internal phase, in which the machine only enters so called 'internal states'. During such an internal phase, the machine can not receive meaningful input at its input port, and does not produce meaningful output. On the other hand, when the machine is in an 'external state', it is open for communication with the environment. To facilitate the constant reception and production of symbols at the input and output port respectively, the symbol λ is fixed, meaning, when appearing at a port, that there is currently no actual symbol at the respective port. The following definitions are due to Verbaan [64]. We slightly modified the definitions for the sake of exposition.

Definition 2.2.1 (Interactive Turing machine (ITM)). *An interactive Turing machine with k work tapes is a 6-tuple $\langle Q, \Sigma, \Gamma, I, q_0, \delta \rangle$, where*

- *Q is a finite set of states;*
- *Σ is a non-empty finite input alphabet;*
- *Γ is a non-empty finite output alphabet;*
- *$I \subset Q$ is a set of internal states;*
- *$q_0 \in (Q - I)$ is the initial state;*
- *$\delta : Q \times \Sigma^k \times (\Sigma \cup \{\lambda\}) \to Q \times \Sigma^k \times \{L, R, none\}^k \times (\Gamma \cup \{\lambda\})$ is a partial transition function.*

Definition 2.2.2 (ITM configuration). *Let $M = \langle Q, \Sigma, \Gamma, I, q_0, \delta \rangle$ be an ITM with k work tapes. A configuration of M is a tuple $\langle q, w_1, ..., w_k, i_1, ...i_k \rangle$, where*

- *$q \in Q$;*
- *$w_j \in \Sigma^*$, for all $1 \leq j \leq k$ is the content of work tape j;*
- *$i_j \in \mathbb{Z}_{\geq 0}$, and $1 \leq i_j \leq |w_j + 1|$, for all $1 \leq j \leq k$ is the position of the tape head on work tape j.*

Definition 2.2.3 (Internal/external configuration). *Let $c = \langle q, w_1, ..., w_k, i_1, ...i_k \rangle$ be a configuration. Then*

$$c \text{ is } \begin{cases} \text{an internal configuration} & \text{if } q \in I \\ \text{an external configuration} & \text{if } q \in (Q - I). \end{cases}$$

Verbaan [64, p. 61] defines an internal phase as follows.

Definition 2.2.4 (Internal phase). *An internal phase is a maximal part of a computation that consists of only consecutive internal configurations.*

Van Leeuwen and Wiedermann, as well as Verbaan, require the interactive Turing machine to produce a non-λ symbol after receiving a non-λ symbol from the environment within finite time. This means that internal phases are required to be finite. Van Leeuwen and Wiedermann [69, p. 209] refer to this condition as "*interactiveness*" or the "*finite delay condition*".

Although technically ITM's receive input streams from $(\Sigma \cup \{\lambda\})^\omega$, the λ's are just a placeholder for 'nothing', and are only interesting in the operational sense. When looking at the translations realised by interactive Turing machines, they are of no importance. Therefore, Verbaan [64, p. 60] proposes to filter the streams in the following way.

Definition 2.2.5 (Filtered streams)**.** *Let x be a stream in $(\Sigma \cup \{\lambda\})^\omega$. We can filter x to a stream x' in Σ^ω by replacing all substrings of the form λ^* in x by empty strings. The same can be done with streams in $(\Gamma \cup \{\lambda\})^\omega$.*

Yet, from an operational point of view, a given machine M can not process just any input stream x that filters to x'. This is because the machine can not accept a symbol other than λ at the input port during internal computation. Whenever the read prefix of the input stream causes the machine to go into an internal state, the input stream should be interleaved with λ's for the duration of M's internal computation phase. Verbaan [64, p. 62] therefore introduces the notion of 'valid input'.

Definition 2.2.6 (Valid input)**.** *We say that a string $x \in \Sigma^\omega$ is a valid input to an ITM M if M can process a string x' that filters to x.*

Now, Verbaan [64] defines what it is for a translation to be realisable by an interactive Turing machine, or 'interactively realisable'.

Definition 2.2.7 (Interactive realisability)**.** *Let M be an ITM. We define the partial translation $\Phi^M : \Sigma^\omega \to \Gamma^\omega$ by letting*

$$\Phi_M(x) = \begin{cases} \textit{The output of } M \textit{ on input } x & \textit{if } x \text{ is a valid input to } M \\ \text{undefined} & \textit{otherwise.} \end{cases}$$

A translation Φ is interactively realisable if there exists an ITM M such that $\Phi = \Phi_M$.

For an account on the properties and complexity of interactively realisable translations, which goes beyond the scope of this thesis, we refer the reader to [64].

Advice

Having introduced the interactive Turing machine model, we now briefly discuss the notion of advice. The concept of advice was first introduced by Karp and Lipton in [31], and plays an important role in non-uniform complexity theory. Non-uniform complexity classes arise when a different algorithm is used for each input size, as opposed to uniform complexity classes, which arise when the same algorithm is used for inputs of all sizes. A thorough account on non-uniform complexity can be found in [2]. A natural model of non-uniform computation is the Boolean circuit, but the Turing machine, basically a uniform model of computation, can be made non-uniform as well by providing it with an advice. Like the more general oracles [57], advice functions provide external, possibly non-Turing-computable information to the machine. Unlike oracles, whose information may depend on the concrete input, the value of an advice function only depends on the length of the machine's input.

Definition 2.2.8 (Advice function)**.** *An advice function is a total function $\alpha : \mathbb{N} \to \Omega^*$, where Ω is a non-empty finite advice alphabet.*

We illustrate the use of advice by an example. This example shows how every standard Turing machine can be transformed into a decider, a machine that halts on all inputs, by feeding it an advice function, whose value's length is exactly the length of the input.

Example 2.2.1. Let M be a Turing machine with finite input alphabet Σ. Without loss of generality, we assume that $\Sigma = \{0, 1\}$ throughout this example. For every $n \in \mathbb{N}$, we construct the advice value $\alpha(n)$ as follows. Let $w_1, w_2, ..., w_{2^n}$ be an enumeration of all strings over Σ of length n. For each string w_i in the enumeration, mark the string if M halts on w_i, and if so, note down the number of computational steps it took the machine to halt on w_i. Let this number be denoted by s_{w_i}. Let $w_{n,max}$ denote the string of length n on which the machine M halts, but in a maximum number of steps. Let $\alpha(n) = w_{n,max}$.

Given an input string x, let M perform its computation as follows. First, M consults its advice for the value $\alpha(|x|) = w_{|x|,max}$. Then, M performs its computation on x and $w_{|x|,max}$ simultaneously. If the machine halts on x within the time it takes to halt on $w_{|x|,max}$, output 1. If not, we know that M does not halt on x, otherwise s_x would have been greater than $s_{w_{|x|,max}}$, which is a contradiction. Therefore, in this case, output 0. Note that on all inputs x' in Σ^*, M outputs either 0 or 1 within $s_{w_{|x'|,max}}$ computational steps.

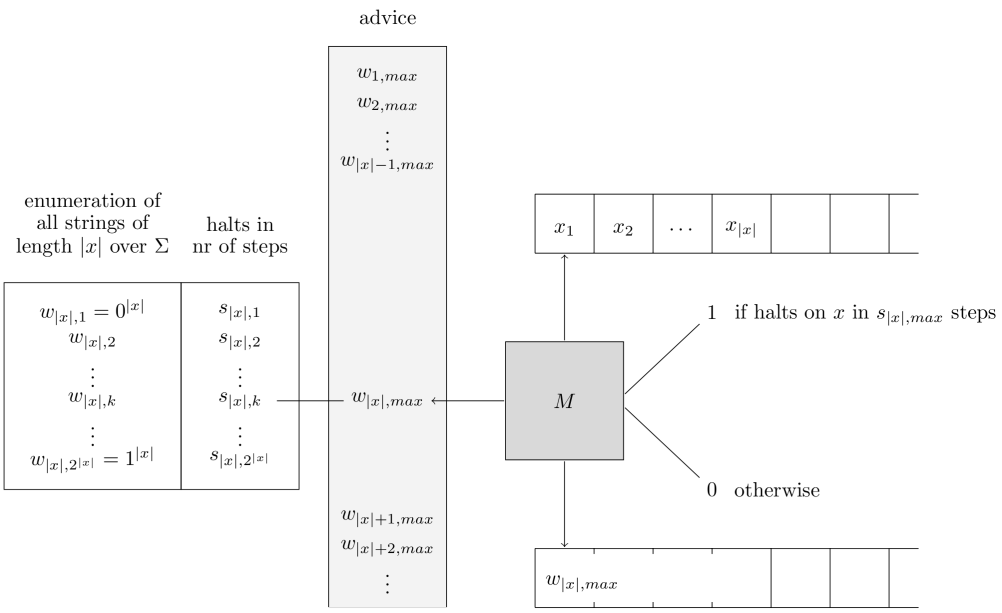


Figure 2.2: Advice function turning Turing machine M into a decider.

Interactive Turing machines with advice

Having discussed the interactive Turing machine and the concept of advice separately, we now discuss a model which combines the two: the interactive Turing machine with advice. The interactive Turing machine with advice is an interactive Turing machine, which, at any time $t > 0$ is allowed to consult its advice for values of at most t. Thereto, the interactive Turing machine with advice uses a special read-only tape, called the advice tape. On receiving the n-th input symbol at the input port, the n-th value of the advice function is automatically appended to the end of the advice tape in one step. The advice tape head can remain where it is during the append operation. The subsequent advice values on the advice tape are separated by a special marker symbol.

In [68, 69] Van Leeuwen and Wiedermann do not provide a formal definition of interactive Turing machines with advice. We present our definition here, based on the informal descriptions provided by the authors.

Definition 2.2.9 (Interactive Turing machine with advice (ITM/A))**.** *An interactive Turing machine with advice and k work tapes is a 6-tuple $M = \langle Q, \Sigma, \Gamma, \Omega, q_0, \delta \rangle$, where*

- *Q is a finite set of states;*
- *Σ is a non-empty finite input alphabet;*
- *Γ is a non-empty finite output alphabet;*
- *Ω is a non-empty finite advice alphabet;*
- *$I \subset Q$ is a set of internal states;*
- *$q_0 \in (Q - I)$ is the initial state;*
- *$\delta : Q \times \Sigma^k \times (\Sigma \cup \{\lambda\}) \times \Omega \to Q \times \Sigma^k \times \{L, R, none\}^k \times \{L, R, none\} \times (\Gamma \cup \{\lambda\})$ is a partial transition function.*

Definition 2.2.10 (ITM/A configuration)**.** *Let $\mathcal{M} = \langle Q, \Sigma, \Gamma, \Omega, q_0, \delta \rangle$ be an ITM/A with k work tapes. A configuration of $\mathcal{M}$ is a tuple $\langle q, w_1, ..., w_k, i_1, ..., i_k, a, m \rangle$, where*

- *$q \in Q$;*
- *$w_j \in \Sigma^*$, for all $1 \leq j \leq$ is the content of work tape j;*
- *$i_j \in \mathbb{Z}_{\geq 0}$, and $1 \leq i_j \leq |w_j + 1|$, for all $1 \leq j \leq k$ is the position of the tape head on work tape j;*
- *$a \in \Omega^*$ is the content of the advice tape;*
- *$m \in \mathbb{Z}_{\geq 0}$ is the position of the advice tape head.*

Definition 2.2.3 of internal and external configurations and internal phases for ITM's also applies to ITM/A's, as does Definition 2.2.6 of valid input.

Having defined interactive Turing machines with advice, we can now define what it means to be computable by an ITM/A. The following definition is a slightly modified version of Verbaan's [64].

Definition 2.2.11 (Non-uniform realisability)**.** *Let M be an ITM/A with advice function α. We define the partial translation $\Phi_{M:\alpha} : \Sigma^\omega \to \Gamma^\omega$ by letting*

$$\Phi_{M:\alpha}(x) = \begin{cases} \textit{The output of } M \textit{ on input } x \textit{ using the advice } \alpha(|x|) & \textit{if } x \text{ is a valid input to } M \\ \text{undefined} & \textit{otherwise.} \end{cases}$$

A translation Φ is non-uniformly realisable if there exists an ITM/A M with an advice function α such that $\Phi = \Phi_{M:\alpha}$.

Van Leeuwen and Wiedermann [69, p. 210] state that interactive Turing machines with advice are "*provably more powerful* than ITM's without advice". A proof of this result can be found in [62].

2.3 Computational power

Van Leeuwen and Wiedermann [68, 69] formalise their claim that a hypercomputational potential can emerge in artificial living systems by showing that the computational power of lineages of cognitive transducers and communities of active cognitive transducers equals the computational power of interactive Turing machines with advice, which are in turn more powerful than standard interactive Turing machines.

Lineages

First, we discuss the computational power of lineages of cognitive transducers. In [69, p. 210], Van Leeuwen and Wiedermann present the following theorem:

Theorem 2.3.1. *A transduction $\phi : \Sigma^\omega \to \Sigma^\omega$ is realised by a lineage of IFTs [lineage of cognitive transducers] if and only if it can be realized by an interactive Turing machine with advice.*

The authors do not prove the theorem in the article, but refer to [62] for a sketch of proof. This proof is very helpful in getting an insight in the idea behind the simulation of ITM/A's by lineages of cognitive transducers, and vice versa. To get a better understanding of the simulation, we present a more detailed version of proof here. To that end, we present and prove two lemma's, that together form the proof for Theorem 2.3.1.

Lemma 2.3.1. *Let $\Phi : \Sigma^\omega \to \Sigma^\omega$ be a translation realised by a lineage of cognitive transducers. Then Φ can be realised by an interactive Turing machine with advice.*

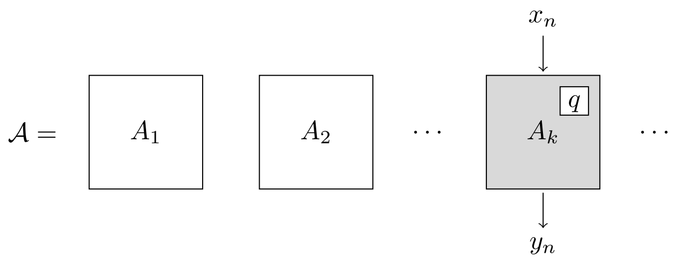
Our proof of lemma 2.3.1 is based on Verbaan [64].

Proof. Let Φ be realised by lineage $\mathcal{A} = \{A_1, A_2, ...\}$, where each cognitive transducer $A_i = \langle Q_i, \Sigma_i, \Gamma_i, q_{0_i}, \delta_i \rangle$. We construct an ITM/A $\mathcal{M}$ and an advice function α, such that $\mathcal{M}$ with α simulates the computation of $\mathcal{A}$, and hence $\Phi_{M:\alpha} = \Phi$. Besides an advice tape, $\mathcal{M}$ has two work tapes, one of which is the so called 'control tape'. Let α be such that $\alpha(k)$ is the encoding of A_k, for all $k \geq 0$.

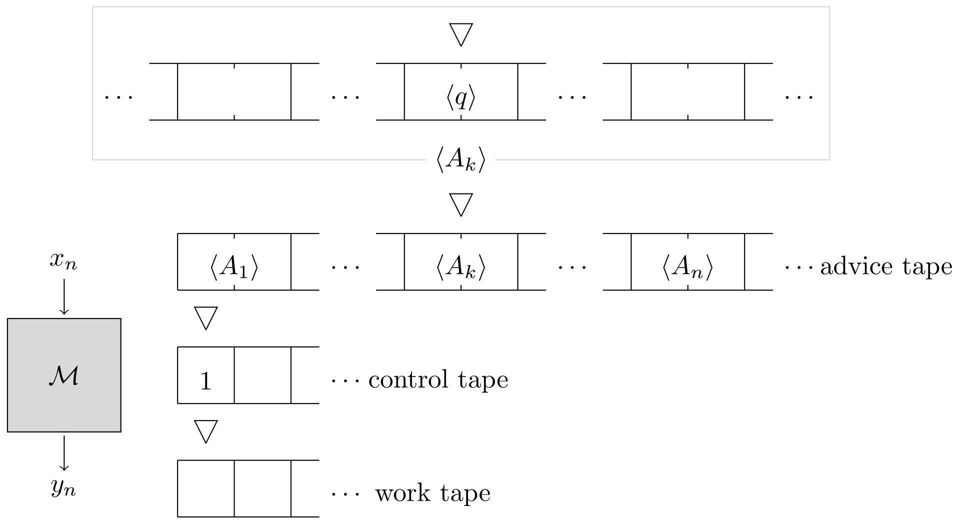
Suppose that after processing prefix $x_1 x_2 ... x_n$ of the input stream x, the controlling cognitive transducer in $\mathcal{A}$ is A_k, and this transducer is in state $q \in Q_k$ (see 2.3). After processing the first n symbols of the input, $\mathcal{M}$'s advice tape contains the string $\alpha(1)\#\alpha(2)\#...\#\alpha(k)\#...\#\alpha(n)$. The head of $\mathcal{M}$'s advice tape is at the beginning of the encoding of q in the encoding of A_k (see 2.3).

On receiving input symbol x_{n+1}, $\mathcal{M}$ continues its computation as follows. First, if the control tape contains a 1, this indicates that $q \in G_k$ and A_k passes over its control to A_{k+1}, who continues computing from state $q \in G_{k+1}$. In this case, $\mathcal{M}$ copies the description of q to its work tape, and moves the head of its advice tape to the description of cognitive transducer A_{k+1}. Using its work tape, $\mathcal{M}$ locates the description of q in $\langle A_{k+1} \rangle$. Otherwise, if the control tape does not contain a 1, indicating that $q \in Q_k - G_k$, this step is skipped.

Then, $\mathcal{M}$ looks up the encoding of the transition tuple $\langle q, x_{n+1}, q', y_{n+1} \rangle$ corresponding to state q and input symbol x_{n+1}. It copies the destination state and the output symbol to the work tape. $\mathcal{M}$ moves its advice tape head to the first symbol of the description of destination state q' in the description of the current automaton. If this state is a global



(a) $\mathcal{A}$ after processing the n-th input symbol, with A_k being the controlling cognitive automaton.



(b) $\mathcal{M}$ after processing the n-th input symbol, with 1 on the control tape.

Figure 2.3: Lineage $\mathcal{A}$ and ITM/A $\mathcal{M}$ after processing the n-th input symbol.

state, i.e. $q' \in G_{k+1}$, then $\mathcal{M}$ writes a 1 on its control tape. Otherwise, the content on the control tape is erased. Finally, $\mathcal{M}$ produces the output symbol y_{n+1} at the output port. □

Lemma 2.3.2. *Let $\Phi : \Sigma^\omega \to \Sigma^\omega$ be a translation realised by an interactive Turing machine with advice. Then Φ can be realised by a lineage of cognitive transducers.*

Our proof of lemma 2.3.2 is based on Verbaan [64], whose proof contains complexity measures, which are not essential for the lemma we want to prove.

Proof. Let Φ be realised by an ITM/A $\mathcal{M}$ with an advice function α. We construct a lineage of cognitive transducers $\mathcal{A} = \{A_1, A_2, ...\}$, such that $\mathcal{A}$ simulates the computation of $\mathcal{M}$ step by step. Let each A_k in $\mathcal{A}$ be such that it simulates $\mathcal{M}$ on input prefixes of length

k. Thereto, A_k has all possible external configurations of $\mathcal{M}$ on an input prefix of length k encoded into its states. Each state in Q_k is a global state.

Suppose that after processing the $(n-1)$-th non-λ input symbol, ITM/A $\mathcal{M}$ is in external configuration $c_n = \langle q, w_1, ..., w_k, i_1, ..., i_k \rangle$, with $q \in (Q - I)$. On receiving the n-th input symbol, the controlling cognitive transducer in $\mathcal{A}$ is A_n. It starts in state $q \in Q_n$, corresponding to $\mathcal{M}$'s external configuration after processing the $(n-1)$-th input symbol. On receiving the n-th input symbol, $\mathcal{M}$ either enters an internal phase, or makes a transition to an external configuration directly. If $\mathcal{M}$ enters an internal phase, during this phase, $\mathcal{M}$ requires only λ's at its input port and produces only λ's at its output port. An internal phase is not simulated by A_n. On the other hand, λ's do not contribute to the computation, and can be safely ignored. Since internal phases are assumed to be finite, eventually $\mathcal{M}$ will reach an external configuration. The whole phase between the two external configurations is simulated by A_n in a single transition. After making this transition A_n passes over control to A_{n+1}. □

Communities

Second, we discuss the computational power of communities of active cognitive transducers. In [69, p. 213] Van Leeuwen and Wiedermann present the following theorem:

Theorem 2.3.2. *Communities of agents [active cognitive transducers] have a computational power equivalent to interactive Turing machines with an advice function whose values grow at most linearly in size with the processing time.*

However, the theorem remains unproven in the article. Instead, the authors refer to [61], where they prove a similar result for the case of an internet machine. An internet machine, which is a model by Van Leeuwen and Wiedermann [61], is quite similar to a community of active cognitive transducers; it is a finite, but time-varying set of simpler machines. In the case of the internet machine, these simpler machines are site machines, which Van Leeuwen and Wiedermann [61] presented to model a personal computer, operated by a human being. According to Van Leeuwen and Wiedermann [69], the internet machine is thus a model of real agents communicating with each other via an Internet-like infrastructure. We adjust the sketch of proof provided by Van Leeuwen and Wiedermann in [62] to the case of communities of active cognitive transducers. This adjustment only works for proving the following lemma.

Lemma 2.3.3. *Let $\Phi : (\Sigma^{|\beta(t)|})^\omega \to (\Sigma^{|\beta(t)|})^\omega$ be a translation realised by a community of active cognitive transducers. Then Φ can be realised by an interactive Turing machine with advice.*

Proof. Let Φ be realised by community of active cognitive transducers $\mathcal{G} = \langle \beta, \Delta, E \rangle$. We construct an ITM/A $\mathcal{M}$ and an advice function α, such that $\mathcal{M}$ with α simulates the computation of $\mathcal{G}$ with E.

First, observe that $\mathcal{G}$ processes and produces streams of symbol packages of varying size, where the size of the package $x_t = \{(x_{i_1,t}, x_{i_2,t}, ..., x_{i_k,t})\}$ depends on the number of community members at time t. The ITM/A, on the other hand, can only receive and produce one symbol at a time via its input and output port respectively. Therefore, we slightly adjust the architecture of the simulating ITM/A, and replace the input and output port by an infinite one-way read-only input and output tape. We assume that the original stream of symbol packages $\{(x_{i_1,t}, x_{i_2,t}, ..., x_{i_k,t})\}_{t=0}^{\omega}$ to $\mathcal{G}$ is translated into a stream processable by the adjusted ITM/A as follows. For consecutive $t = 1, 2, ...$, the symbol package $x_t = \{(x_{i_1,t}, x_{i_2,t}, ..., x_{i_k,t})\}$ is written on the input tape in the form of a so called input 'block' $x_{i_1,t}x_{i_2,t}...x_{i_k,t}$. Subsequent blocks are separated by a special marking symbol #. Outputs are written in a similar block-wise manner on the output tape.

Let the advice function α be such that $\alpha(k) = \langle\beta(k), \Delta(k)\rangle$. For the sake of exposition, we assume $\mathcal{M}$ has two advice tapes: the description tape where the values of β are written, and the timing tape, where the values of Δ are written. Moreover, we assume $\mathcal{M}$ has two work tapes: the configuration tape, and the communication tape.

Suppose that after processing the first n symbol packages of the input stream, the configuration of the community is $\boldsymbol{c_n} = \langle\{c_1, ..., c_k\}, X\rangle$, where c_i denotes the current configuration of active cognitive transducer M_i in the community that that time, for each $1 \leq i \leq k$, and $\{\langle i_1, \langle M_{i_1}\rangle\rangle, \langle i_2, \langle M_{i_2}\rangle\rangle, ..., \langle i_k, \langle M_{i_k}\rangle\rangle\} = \beta(n)$. After processing the first n input blocks, $\mathcal{M}$'s configuration tape contains for each $\langle i, \langle M_i\rangle\rangle \in \beta(n)$ the id i followed by the encoding $\langle c_i\rangle$ of M_i's current configuration. The communication tape contains the current contents X of the grid environment.

On receiving input package x_{n+1}, ITM/A $\mathcal{M}$ continues its computation as follows. First, $\mathcal{M}$ goes into an internal state. It consults its advice for Δ. ITM/A $\mathcal{M}$'s timing tape's head is at the beginning of the encoding of $\Delta(n + 1)$, reading the first id i_1 of the cognitive transducers in the community that are updated at this time. In general, when reading the id i_j on the advice tape, on its configuration tape $\mathcal{M}$ looks up the current configuration $c_{i_j} = \langle q_{i_j}, p_{i_j}\rangle$ of M_{i_j}. On its communication tape, $\mathcal{M}$ looks up the symbol π at M_{i_j}'s current location p_{i_j}. Next, on its description tape, $\mathcal{M}$ looks up the encoding of the transition tuple $\langle q_{i,j}, \pi, x_{i_j,t+1}, q'_{i_j}, \pi'_{i_j}, m, y_{i_j}\rangle$ in the encoding $\langle M_{i_j}\rangle$. On its configuration tape, $\mathcal{M}$ overwrites the current state q_{i_j} of M_{i_j} with the new state q'_{i_j}. Then, $\mathcal{M}$ overwrites the cell under its communication tape head with the symbol π', and moves its communication tape head according to m. Finally, $\mathcal{M}$ writes the output symbol y_{i_j} to the output tape.

Then, $\mathcal{M}$ moves its timing tape's head to the next id on the timing tape, and proceeds with the simulation of the next active cognitive transducer, until it reads a # on the timing tape. Finally, $\mathcal{M}$ enters an external state again. □

The reverse comparison, which must also hold to prove Theorem 2.3.2, can not be directly deduced from the case of the Internet machine. This is because Van Leeuwen and Wiedermann's proof relies on the fact that a single site machine operated by a real agent, whose role it is to supply the values of the advice function, can simulate an interactive Turing machine with advice. This is not the case for a single active cognitive transducer.

Chapter 3
Hypercomputation

In the previous chapter, we discussed Van Leeuwen and Wiedermann's [68, 69] claim that a hypercomputational potential can emerge in artificial living systems. In order to understand Van Leeuwen and Wiedermann's claim, as well as to place their contribution in a broader scientific debate, we discuss the notion of hypercomputation in this chapter.

According to Copeland [17, p. 251], "[h]ypercomputation is the computation of functions or numbers that cannot be computed in the sense of Turing". He refers to hypercomputers as machines that "compute functions or numbers, or more generally solve problems or carry out tasks, that lie beyond the reach of the standard universal Turing machine" [17, p. 251]. Stannett [52, p. 115] refers to a hypercomputational system as "one that 'computes' non-computable behaviours", where the term 'non-computable' means 'not computable by Turing machines'. Stannett [52] makes a distinction between 'non-Turing machines' and 'super-Turing machines'. The former refers to all those machines that "can behave in hypercomputational ways" [52, p. 116]. Non-Turing machines may include machines whose computations are incomparable to those of standard Turing machines. On the other hand, the term 'super-Turing machines' refers to those machines that, besides being able to behave hypercomputationally, can also simulate standard Turing machines. Therefore, super-Turing machines can be said to be more powerful than Turing machines.

As is clear from the above definitions, the notion of being computable by Turing machines is central to hypercomputation. Therefore, we first discuss Turing-computability in Section 3.1. Then, we discuss a more general notion of computability — relative computability — of which Turing-computability is a special instance. A more general notion of computability is advocated by Copeland and Sylvan [19], who argue that *all* computation is performed relative to some set of resources or primitive capabilities. An interesting question is what resources can lead to hypercomputational power. Ord [42] has made an assessment of the resources used by a selection of hypercomputational models from the literature. In Section 3.2, we discuss the resources that are relevant to Van Leeuwen and Wiedermann's [68, 69] models: non-Turing-computable information sources, infinite operation, interaction, evolution, asynchrony, and infinite specification. In Section 3.3, we discuss a framework for measuring and comparing the relative power of hypercomputational models. We extend Ord's [42] assessment of the power of several hypercomputational models by including Van Leeuwen and Wiedermann's [68, 69] models. Although the study of theoretical models of hypercomputation gives insight in the resources required for a model to gain hypercomputational power, the question remains whether these hypercomputational models can be physically realised. Another question is whether the hypercomputational power of hypercomputational systems can be exploited for solving given non-Turing-computable problems. In Section 3.4, we discuss physical realisability, as well as exploitability of hypercomputation.

3.1 Computability

Hypercomputation is computation of functions that are not 'computable' by Turing machines. To understand hypercomputation, one should therefore know what it is to be, or not to be computable by a Turing machine. The notion of 'computability' is a central subject of computability theory, a branch of theoretical computer science. It is concerned with the question of what is, and what is not computable.

Computability is often explicitly defined for functions $f \subseteq \Sigma^* \to \Sigma^*$ on the set of finite strings over an arbitrary finite alphabet Σ. Computation can also be defined for functions on other sets M, such as the set of natural numbers $\mathbb{N}$. Finite words in Σ^* are then used as 'names' for elements of M [67]. Each natural number $n \in \mathbb{N}$, for example, can be represented in unary as the string 1^n, which is the string of n times the symbol 1. In computable analysis [67], a branch of computability theory, computability is defined for functions $f \subseteq \Sigma^\omega \to \Sigma^\omega$ on the set of infinite streams over an arbitrary finite alphabet Σ. This set Σ^ω has continuum cardinality, and can therefore be used as a set of names for other sets M with the same cardinality, such as the set of real numbers $\mathbb{R}$. In the examples provided in this chapter, we assume that Σ is a fixed alphabet containing the symbols 0 and 1. Moreover, computability can be defined on elements of Σ^* and Σ^ω. A finite string $w \in \Sigma^*$ is computable if and only if the constant function $f : \{\varepsilon\} \to \Sigma^*$ with $f(\varepsilon) = w$ is computable, where *varepsilon* denotes the empty string. Likewise, an infinite stream $y \in \Sigma^\omega$ is computable if and only if $f : \{\varepsilon\} \to \Sigma^\omega$ with $f(\varepsilon) = y$ is computable [67].

Before being able to investigate what functions are or are not computable, one should first have a clear and formal definition of what one intuitively understands by 'computing' and a 'computation'.

Turing-computability

The most well-known notion of computation is the notion that refers to the calculation of mathematical functions, according to an 'algorithm'. Soare refers to this notion of computation as "a process whereby we proceed from initially given objects, called *input*, according to a fixed set of rules, called a *program*, *procedure*, or *algorithm*, through a series of *steps* and arrive at the end of those steps with a final result, called the *output*" [51, p. 6]. An algorithm, according to Rogers [26, p. 1], is "a clerical (i.e. deterministic, book-keeping) procedure which can be applied to any of a certain class of symbolic *inputs* and which eventually yield, for each such input, a corresponding symbolic *output*". This notion of computation is also referred to as 'effective computation', 'algorithmic computation', or 'mechanistic computation'. In what follows, we will use the term 'effective computation'. It is this notion of effective computation and effective computability which was formalised by Turing [56].

In an article published in 1937, Turing [56] introduced a conceptual model for effective computation: the computing machine, which has come to be known as the Turing machine (TM). The Turing machine is a formal model of an idealised human computing agent — a 'computor' — calculating real numbers using a pencil, a rubber, and a potentially unlimited amount of paper, according to a set of rules, but without insight. Where the human computor is supplied with paper, the Turing machine is supplied with a potentially infinite tape, consisting of a number of squares, each capable of bearing a symbol from a finite alphabet. The machine has a tape head, which can read the symbol in the square it is currently on. The machine can enter a finite number of states. Its possible behaviour is at any moment determined by its current state and read symbol. The machine has a small repertoire of elementary operations. First, it can erase the symbol in the observed square, or write down a symbol in the observed square if the respective square is blank. Second, it can change the square being observed by moving its tape head one square to the left or to the right. Finally, the machine can change its state. Turing argues that "these operations include all those which are used in the computation of a number" [56, p. 232]. In Turing's definition, "a number is computable if its decimal can be written down by a machine" [56, p. 230].

Instead of producing real numbers digit by digit, in modern definitions Turing machines often compute functions from $f \subseteq: \Sigma^* \to \Sigma^*$. First, we give a formal definition of a Turing machine.

Definition 3.1.1 (Turing machine). *A Turing machine (TM) is a 6-tuple $\langle Q, \Sigma, \Gamma, q_0, F, \delta \rangle$, where*

- *Q is a finite set of states;*
- *Σ is a non-empty finite input alphabet;*
- *$\Gamma \supseteq \Sigma \cup \{\sqcup\}$ is a non-empty finite tape alphabet, where $\sqcup$ is a special blank symbol;*
- *$q_0 \in Q$ is the initial state;*
- *$F \subseteq Q$ is a set of halting states;*
- *$\delta : Q \times \Sigma \to \mathbb{Q} \times \Gamma \times \{L, R, none\}$ is a partial transitions function.*

Prior to the start of the computation, a finite input string $x \in \Sigma^*$ is inscribed on the Turing machine's tape. The Turing machine starts in its special initial state q_0. Then, the Turing machine makes state transitions according to its transition function δ, thereby manipulation the string on its tape. If the Turing machine reaches one of its special halting states, the computation stops and the string on the tape is the resulting output. If the Turing machine never halts on its input, it means it is in an infinite loop, and does not produce an output. Then the function is not defined on that specific input. Let $\phi_T(x)$ denote the output of TM T on input x. Then each Turing machine T computes a single partial function ϕ_T. Turing identified the 'computable' numbers as those that can be computed by a Turing machine. To avoid confusion, we refer to this notion of computability as 'Turing-computability', and in general use the bare term 'computable' relative to some specific model of computation. We rephrase Turing-computability in terms of functions.

Definition 3.1.2 (Turing-computability). *A function f is Turing-computable if and only if there exists a Turing machine T such that $f = \phi_T$.*

The thesis that the functions that are intuitively effectively computable are exactly the Turing-computable functions has come to be known as Turing's thesis.

Thesis 3.1.1 (Turing's thesis). *Every function which is intuitively effectively computable is Turing-computable.*

Turing's thesis is a claim that can not be proven, because the underlying notion of effective procedure remains an intuitive one. On the other hand, Turing's thesis can be regarded as a kind of definition of effective computability. Actually, there exist many different interpretations of Turing's thesis. Goldin and Wegner [24] point out that a common interpretation of Turing's thesis is the claim that *all* kinds of computation, not just the effective computation of mathematical functions, can be accurately modelled by Turing machines. Goldin and Wegner [24], among others, argue that this is in fact a misinterpretation of the thesis. For now, we stress that the scope of Turing's thesis is limited to the intuitive notion of effective computability.

Turing showed that "although the class of computable numbers [or functions] is so great, and in many ways similar to the class of real numbers, it is nevertheless enumerable" [56, p. 230]. We do not go into detail on how to show this. We just assume that there exists a total Turing-computable surjection from $\mathbb{N}$ to the set of Turing machines— an encoding — such that every $e \in \mathbb{N}$ is the index of exactly one Turing machine T, or of the empty Turing machine, and every Turing machine T is encoded into a number $e \in \mathbb{N}$. We refer to the encoding of a Turing machine T as $\langle T \rangle$. Then we denote the e-th Turing-computable function by ϕ_e, and $\phi_e = \phi_T$ if $e = \langle T \rangle$. For an example using a Gödel numbering we refer to [12].

After introducing the Turing machine, Turing even went a step further and introduced the universal Turing machine (UTM), which takes as input an index e of a Turing machine, as well as an input string x and simulates the e-th TM on input x. The universal Turing machine can be seen as the early conceptual model of general purpose computers. In the theoretical context, the universal Turing machine can compute all Turing-computable functions.

The Turing machine has been, and still is, the paradigmatic model of effective computation. Numerous other models of effective computation have been shown equally powerful as the Turing machine. Among these models are combinatory logic [49], the λ-calculus [9, 10], general recursive functions [23], as well as more recent models, such as cellular automata [65, 11]. When we speak of 'computable by Turing machines', we thus implicitly include 'computable by every Turing-equivalent model of computation'. The thesis stating that the functions that are effectively computable according to the intuitive notion are exactly the Turing-computable, general recursive, or λ-definable functions, or those computable by equivalent models of computation, is known as the Church-Turing thesis.

So far, we have only discussed the Turing machine as a model positively identifying those functions that are effectively computable. Interestingly, Turing [56] used his Turing machine model to show that there exist functions that are not effectively computable. Actually, since there exist uncountably many functions from Σ^* to Σ^* and only countably many of these functions are Turing-computable, this leaves uncountably many functions non-Turing-computable. A famous example of such a non-Turing-computable function is the halting function, which is defined as follows.

Definition 3.1.3 (Halting function)**.** *The halting function h is the function given by*

$$h(e,x) = \begin{cases} 1 & \textit{if } \phi_e(x) \textit{ is defined (alternatively: TM with index } e \textit{ halts on input } x\textit{)} \\ 0 & \textit{otherwise.} \end{cases}$$

Closely related to the halting function is the halting set.

Definition 3.1.4 (Halting set)**.** *The halting set K_0 is defined as*

$$K_0 = \{\langle e, x\rangle \mid \text{ the } e\text{-th Turing machine halts on input } x\}.$$

Theorem 3.1.1. *The halting function is not Turing-computable.*

Proof. Suppose that the halting function h is Turing-computable. In other words, suppose there exists a Turing machine H that computes h. Now we construct a new machine H', which on input e first simulates H on $tuplee, e$ as a subroutine on the input, and goes into an infinite loop if H outputs 1 (if the e-th Turing machine T halts on input e), and outputs 1 if H outputs 0 (if the e-th Turing machine T does not halt on input e). Now consider what happens when H' is fed input $\langle H' \rangle$. First assume that H' halts on $\langle H' \rangle$, and therefore its subroutine H outputs 1. Then by definition of H', H' goes into an infinite loop, and does not halt. On the other hand, assume that H' does not halt on input $\langle H' \rangle$, and therefore H''s subroutine H outputs 0. Then by definition of H', H' halts and outputs 1. In both cases, we derive a contradiction. □

The existence of non-Turing-computable functions has motivated the study of models that actually do compute the non-Turing-computable. In other words, it motivated the study of hypercomputation. On the one hand, would one find a model of effective computation that computes a non-Turing-computable function, Turing's thesis would be proven false. On the other hand, hypercomputational models do not necessarily violate Turing's thesis, since they might compute in a non-effective way.

Computability, decidability, and enumerability

Before we discuss a more general view of computation and computability, we briefly discuss two other important concepts in computability theory: decidability, and enumerability. Computation usually concerns the calculation of functions, and computability the existence of a procedure to perform the computation of a given function. Decidability, on the other hand, concerns the existence of a procedure for solving a given 'decision problem'. A decision problem is a problem of the form "is x an element of A?", where A is a set. The answer to a decision problem is either 'yes' or 'no', or 1 or 0 respectively. Semi-decidability is the existence of a procedure that answers 1 if the element is indeed an element of the set, and doesn't give an answer otherwise. Enumerability is about the existence of a procedure for enumerating the elements of a given set in any order, possibly with duplicates. As in the case of computability, the most well-known notions of decidability and enumerability are effective decidability and effective enumerability. As we will see now, the effective computability of functions, the effective decidability of sets and the effective enumerability of sets are closely related.

The problem of deciding membership of a set A can be formulated in terms of the problem of computing A's characteristic function χ_A.

Definition 3.1.5 (Characteristic function)**.** *The characteristic function χ_A of a set A is defined by*

$$\chi_A(x) \begin{cases} 1 & \textit{if } x \in A \\ 0 & \textit{if } x \notin A. \end{cases}$$

Using the characteristic function, effective decidability can be defined in terms of Turing-computability as follows.

Definition 3.1.6 (Effective decidability). *A set $A \subseteq \mathbb{N}$ is effectively decidable if and only if its characteristic function χ_A is Turing-computable.*

A set is thus effectively decidable if and only if there exists a Turing machine that computes its characteristic function. Characteristic functions are total functions, and Turing machines computing these functions halt on all inputs. Such Turing machines are called 'deciders'. In general, Turing machines compute partial functions. The partial variant of a set's characteristic function is its semi-characteristic function.

Definition 3.1.7 (Semi-characteristic function). *The semi-characteristic function S_A of a set A is defined by*

$$S_A(x) \begin{cases} 1 & \text{if } x \in A \\ \text{undefined} & \text{if } x \notin A. \end{cases}$$

The effective semi-decidability of a set A can be formulated in terms of the effective computability of its semi-characteristic function S_A.

Definition 3.1.8 (Effective semi-decidability). *A set $A \subseteq \mathbb{N}$ is effectively semi-decidable if and only if its semi-characteristic function S_A is Turing-computable.*

On the other hand, every function from Σ^* to Σ^* can be turned into the problem of deciding the graph associated with the function.

Definition 3.1.9 (Graph). *Let $f : \Sigma^* \to \Sigma^*$ be a partial function, where Σ is some finite non-empty alphabet. Then the graph of f is defined as*

$$\text{Graph}(f) = \{\langle x, y\rangle \in \Sigma^* \times \Sigma^* \mid f(x) = y\}.$$

Proposition 3.1.1. *A total function f is Turing-computable $\Leftrightarrow$ Graph(f) is effectively decidable.*

Proof. ($\Rightarrow$) Suppose $f : \mathbb{N} \to \mathbb{N}$ is a total Turing-computable function. Then there is a Turing machine T, such that $\phi_T = f$ and T halts on all inputs. Now we can design a Turing machine M which for all $\langle x, y\rangle$ simulates T on input x and outputs 1 if $\phi_T(x) = y$, and outputs 0 otherwise. This Turing machine M decides Graph(f).

($\Leftarrow$) Suppose Graph(f) is effectively decidable. Then there is a Turing machine T computing the characteristic function $\chi_{\text{Graph}(f)}$. Now we can design a Turing machine M which on input x simulates T sequentially on $\langle x, 1\rangle, \langle x, 2\rangle, ...$ until T outputs a 1 on input $\langle x, y\rangle$. Since f is assumed to be a total function, this will happen eventually. Then M outputs y. Turing machine M computes f.

□

Proposition 3.1.2. *A partial function f is Turing-computable $\iff$ Graph(f) is effectively semi-decidable.*

Proof. ($\Rightarrow$) Suppose $f : \mathbb{N} \to \mathbb{N}$ is a partial Turing-computable function. Then there is a Turing machine T, such that $\phi_T = f$. Now we can design a Turing machine M which for all $\langle x, y\rangle$ simulates T on input x, and outputs 1 if T halts on x and $\phi_T(x) = y$. Otherwise, M does not halt. This Turing machine M semi-decides Graph(f).

(⇐) Suppose Graph(f) is effectively semi-decidable. Then there is a Turing machine T computing the semi-characteristic function $S_{\text{Graph}(f)}$. Now we can design a Turing machine M which on input x simulates T in parallel on $\langle x, 1\rangle, \langle x, 2\rangle, ...$ using the dovetailing technique. If T halts on input $\langle x, y\rangle$ and outputs 1, then M outputs 1. Otherwise, M does not halt. This Turing machine computes the partial function f.

□

A set A is enumerable if there exists a process for enumerating the members of A. Effective enumerability is related to Turing-computability in the following way:

Definition 3.1.10 (Effective enumerability)**.** *A set A is effectively enumerable if and only if $A = \emptyset$ or there is a total Turing-computable function f such that A is* $\{f(0), f(1), f(2), ...\} = \text{range}(A)$.

Enumerability is sometimes called semi-decidability, because a set A is effectively enumerable if and only if the semi-characteristic function of A is Turing-computable. A set A is called co-effectively enumerable if its complement $\overline{A}$ is effectively enumerable. Moreover, if both A and $\overline{A}$ are effectively enumerable, A is also effectively decidable.

Relative computability

In the above, we have mainly focussed on one particular, very important notion of computability: effective computability. Now it is time to take a more general view of computation and computability again and discuss relative computability. Copeland and Sylvan [19, p. 46] argue that "[c]omputability is a relative notion, not an absolute one" and "[t]he extent of the computable functions is *resource*-relative". They add that "[a]ll computation, classical or otherwise, takes place relative to some set or other of primitive capabilities: *all* computation is relative computation" [19, p. 47]. Copeland and Sylvan admit that the Turing-computable functions are of special interest because these are the functions that are computable by an idealised human mathematician. On the other hand, they point out that another interesting set of functions is the set of functions that are in principle implementable in the real world. They add that this set of functions may or may not coincide with the set of Turing-computable functions. Taking a more general view of computation and computability is therefore relevant to our discussion of hypercomputation. Moreover, relative computability theory gives insight in the different levels of hypercomputational power by means of the arithmetical hierarchy, which we discuss in Section 3.3.

Turing's o-machine [57] is a model of relative computation. An o-machine is a Turing machine that can be supplied with some possibly non-algorithmic information in the form of an 'oracle'. An oracle is usually represented as a subset of the natural numbers $A \subseteq \mathbb{N}$. Beside the primitive operations performable by a standard Turing machine, an o-machine can ask its oracle questions of the form "is n in A?", to which the oracle responds directly with 1 if n is indeed a member of A, or with 0 otherwise. One can also see it as if an o-machine with oracle A is a Turing machine equipped with the primitive capability to solve the decision problem associated with A. It is also possible to formulate oracles in terms of functions from Σ^* to Σ^*. An o-machine with an oracle function g can ask its oracle questions of the form "what is $g(m)$?", to which the oracle responds directly with the value of $g(m)$

if $g(m)$ is defined and with "undefined" otherwise. In order to ask questions to the oracle, and to receive answers, the o-machine is equipped with a special query tape.

To illustrate the power of oracles, we describe how every effectively enumerable set can be decided by an o-machine with the halting set K_0 as its oracle set.

Example 3.1.1. As a trivial case, consider the halting set, which is effectively enumerable. Let M be an o-machine computing the characteristic function of its oracle. Then with oracle K_0, o-machine M computes the halting function. In the general case, consider a recursively enumerable set B. From Definition 3.1.8, we know that since B is recursively enumerable, there is a Turing machine T computing S_B. By definition, T will halt on input x and output 1 if $x \in B$ and will not halt otherwise, on all inputs x. On input x, the o-machine M deciding B asks its oracle whether $\langle\langle T\rangle, x\rangle \in K_0$. If this is the case, indicating that T halts on x, and therefore $x \in B$, the o-machine outputs 1. If this is not the case, the M outputs 0. Note that M halts on all inputs x. Besides the recursively enumerable sets, o-machines with the halting set as oracle also decide the co-recursively enumerable sets. This computation proceeds in the same way as that described above, except that the o-machine outputs 0 instead of 1 and vice versa.

It's important to note that an o-machine itself is a finite object, which has no oracle associated to it, but can use any oracle which may be provided to it. In fact, the functions computed by o-machines have two arguments: an oracle and an input string. Therefore, technically they compute a functional — a mapping whose arguments may themselves be number-theoretic functions or sets — rather than a function. In the following definition, we make use of the fact that o-machines can be enumerated in the same way as standard Turing machines.

Definition 3.1.11 (The e-th partial computable functional)**.** *The e-th partial computable functional — Φ_e — is the functional computed by the e-th o-machine M_e. Then Φ_e^A denotes the e-th functional acting on oracle set A, and Φ_e^g denotes the e-th functional acting on oracle function g.*

The o-machine gives rise to the following definition of relative Turing-computability of functions and Turing-reducibility of sets.

Definition 3.1.12 (Relative Turing-computability)**.**

- *A partial function f is A-Turing-computable if f is computable by an o-machine with oracle set A.*
- *A partial function f is g-Turing-computable, written $f \leq_T g$, if f is computable by an o-machine with oracle function g.*
- *A set B is A-Turing-decidable or Turing-reducible to A, written $B \leq_T A$, if the characteristic function of B — χ_B — is A-Turing-computable.*

Relative Turing-computability gives rise to a degree structure, called the Turing universe. We begin with defining the Turing-equivalence relation.

Definition 3.1.13 (Turing-equivalence)**.** *Let $A, B \subseteq \mathbb{N}$. We say A is* Turing-equivalent *to B (write $A \equiv_T B$) if $A \leq_T B$ and $B \leq_T A$.*

Now, we can define the notion of a Turing degree, also called ‘degree of unsolvability’, based on [12].

Definition 3.1.14 (Turing degree)**.**

- *The* Turing degree *of a set $A \subseteq \mathbb{N}$ is defined to be*

$$\deg(A) =_{def} \{X \subseteq \mathbb{N} | X \equiv_T A\}.$$

- *The* Turing degree *of a function $f : \Sigma^* \to \Sigma^*$ is defined to be*

$$\deg(f) =_{def} \deg(\text{Graph}(f)).$$

The Turing degree of a set A can be understood as the equivalence class of a set A — that is, $[A]_{\equiv_T}$ — or a function f with respect to Turing-equivalence. The Turing degree of a set A shows the class of problems that becomes solvable given solutions to a certain problem A. Formulated differently, a Turing degree gathers sets, binary reals, or functions which are “computationally indistinguishable from each other, in the sense that they are mutually Turing computable from each other” [14, p. 1356]. The set of all computable sets has Turing degree $\mathbf{0} = \deg(\emptyset)$. This is also the least Turing degree. With the definition of Turing degrees in place, we define the Turing universe.

Definition 3.1.15 (Turing universe)**.** *The Turing universe $\mathcal{D}$ is the set $2^{\mathbb{N}}/\equiv_T$ — that is, the set of all Turing degrees. We define a partial ordering $\leq$ induced by $\leq_T$ on $\mathcal{D}$ by*

$$\deg(B) \leq \deg(A) \Leftrightarrow_{def} B \leq_T A.$$

The set $\mathcal{D}$ is uncountable, and there is no greatest member of $\mathcal{D}$.

Definition 3.1.16 (Halting set of the e-th Turing machine)**.** *The halting set of the e-th Turing machine is defined as $W_e = \text{dom}(\phi_e)$, for each $e \in \mathbb{N}$. We relativise W_e to subsets $A \subseteq \mathbb{N}$, such that $W_e^A = \text{dom}(\Phi_e^A)$, for each $e \in \mathbb{N}$.*

Next, we define the jump, which is a generalisation of the halting set K_0 to o-machines with oracle A. The definitions are based on [12].

Definition 3.1.17 (Jump)**.** *The jump A' of a set A is defined as*

$$A' =_{def} \{\langle x, y\rangle \mid x \in W_y^A\} = K_0^A.$$

The $(n+1)$-th jump of A is defined as

$$A^{(n+1)} =_{def} (A^{(n)})'.$$

The jump $\boldsymbol{a'}$ of a degree $\boldsymbol{a}$ is defined as

$$\boldsymbol{a'} =_{def} \deg(A').$$

The $(n+1)$-th jump of a degree $\boldsymbol{a}$ is defined as

$$\boldsymbol{a}^{(n+1)} =_{def} (\boldsymbol{a}^{(n)})' =_{def} \deg(A^{(n+1)}).$$

3.2 Resources

In the previous section, we discussed a general notion of computability: relative computability. A relative view of computability is advocated by Copeland and Sylvan [19], who argue that all computation takes place relative to a set of resources or primitive capabilities. Some resources, such as extra work tapes, multiple tape heads, non-determinism and synchronous parallelism, may lead to more efficiency, but do not lead to an increase in computational power when added to a standard Turing machine. A central question for hypercomputation is which resources do actually lead to hypercomputational power.

Ord [42] identifies a number of resources used by a variety of hypercomputational models from the literature. Among these resources are infinite memory, which Ord points out is not in itself sufficient for hypercomputation, non-Turing-computable information sources, infinite specification, infinite computation and fair non-determinism. Ord [42, p. 44] concludes his study by adding that "[i]t would also be of considerable interest to examine the more 'realistic' resources of interaction an infinite run-time as discussed by Jan van Leeuwen and Jirí Wiedermann", referring to Van Leeuwen and Wiedermann's article [61]. Actually, Van Leeuwen and Wiedermann [61] argue that it is the *simultaneous* use of the resources infinite operation, interaction, and non-uniform evolution which leads to their models' hypercomputational potential. In this section, we discuss a selection of resources relevant to our discussion of Van Leeuwen and Wiedermann's [68, 69] lineages of cognitive transducers and communities of active cognitive transducers: non-Turing-computable information sources, infinite operation, interaction, non-uniform evolution, asynchrony and infinite specification.

Non-Turing-computable information sources

Oracles and advice functions are external information resources which can provide a computational model with possibly non-Turing-computable information. Although there is no reason to believe that Turing [57] actually proposed the o-machine as a hypercomputational model, the model is brought into the realm of hypercomputation by Copeland [16]. In Example 3.1.1, we have shown that o-machines with the universal halting set K_0 as oracle can solve the non-Turing-computable halting function. An advice function is a special kind of oracle, whose value does not depend on the concrete input, but rather on the size of the input. Advice functions are functions $\alpha : \mathbb{N} \to \Omega^*$, whereas oracle functions are functions $g : \Omega^* \to \Omega^*$, where Ω denotes a non-empty advice or oracle alphabet. In Example 2.2.1, we have shown that a Turing machine with advice with linear advice can solve the halting function.

Here, we show that as a resource advice is as powerful as oracles. More specifically, we show that the set of functions computable by Turing machines with advice coincides with the set of functions computable by o-machines, relative to comparable advice and oracle functions. This will be useful in measuring the hypercomputational power of Van Leeuwen and Wiedermann's models later on in Section 3.3.

Theorem 3.2.1. *Let Φ be a partial functional. Then $\Phi = \Phi_M$ for some o-machine M $\iff$ $\Phi = \Phi_{M'}$ for some Turing machine with advice M', in such a way that for all oracle functions g, $\Phi^g_M = f \Rightarrow \Phi^{\alpha^g}_{M'} = f$, where α^g is an advice function comparable to g and for all advice functions α, $\Phi^{\alpha}_{M'} = \Phi^{g^\alpha}_M = f$, where g^α is an oracle function comparable to α.*

First, we introduce the notion of 'comparable advice' and 'comparable oracle'.

Definition 3.2.1 (Comparable advice). *Let $g : \Sigma^* \to \Sigma^*$ be an oracle function. First, observe that for all $n \in \mathbb{N}$, the number of strings over Σ of length n is $|\Sigma|^n$. Let $\{w_{n,1}, w_{n,2}, ..., w_{n,|\Sigma|^n}\}$ denote the set of strings over Σ of length n. The cardinality of this set increases as n increases, but it is finite for all n. We construct the comparable advice $\alpha^g : \mathbb{N} \to \Sigma^*$ such that $\alpha^g(n)$ is the concatenation of all $|\Sigma|^n$ oracle values of arguments of length n, for all n. In other words: $\alpha^g(n) = g(w_{n,1})\#g(w_{n,2})\#...\#g(w_{n,|\Sigma|^n})$, for all n.*

For the other way around, we introduce the notion of a 'comparable oracle'.

Definition 3.2.2 (Comparable oracle). *Let $\alpha : \mathbb{N} \to \Sigma^*$ be an advice function. We construct the comparable oracle function g^α such that $g\alpha$ maps all arguments of length n to the same value $\alpha(n)$, for all n. In other words, $g^\alpha(w_{n,1}) = g^\alpha(w_{n,2}) = ... = g^\alpha(w_{n,|\Sigma|^n}) = \alpha(n)$.*

Now, we prove Theorem 3.2.1 by proving Lemma 3.2.1 and Lemma 3.2.2.

Lemma 3.2.1. *Suppose $\Phi = \Phi_M$ is a functional computable by a o-machine M. Then there is a Turing machine with advice M' such that for all oracle functions g, $\Phi^{\alpha^g}_{M'} = \Phi^g_M$, where α^g is the advice comparable to g.*

Proof. Let $\Phi = \Phi_M$ be the partial functional computed by o-machine M. Suppose M acts on oracle function g and let M' act on advice function α^g. TM/A M' works just like o-machine M, but every time M consults its oracle for the value $g(x)$, M' consults its advice for the value of $\alpha^g(|x|)$. This advice value contains the oracle values for *all* inputs of size $|x|$, including that of x itself. Therefore, M' moves its advice tape head to the oracle value corresponding to x. Then, M' continues its computation like M. □

Lemma 3.2.2. *Suppose $\Phi = \Phi_{M'}$ is a functional computed by a Turing machine with advice M'. Then there is an o-machine M such that for all advice functions α, $\Phi^{g^\alpha}_M = \Phi^\alpha_{M'}$, where g^α is an oracle comparable to α.*

Proof. Let $\Phi = \Phi_M$ be the partial functional computed by TM/A M'. Suppose M' acts on advice function α and let M act on oracle function g^α. O-machine M works just like TM/A M'. Every time M' consults its advice for the value $\alpha(|x|)$, M consults its oracle for the value $g^\alpha(x)$. Then, M continues its computation like M'. □

Now in a sense we can say that if a function f is computable by a Turing machine with advice α, then f is computable by an o-machine with comparable oracle function g^α. Hence, f is Turing-reducible to g^α. More formally

$$f \text{ is computable by an Turing machine with advice } \alpha \Rightarrow f \leq_T g^\alpha.$$

On the other hand, if a function f is computable by an o-machine with oracle function g, and hence $f \leq_T g$, then f is computable by a Turing machine with advice α^g. More formally

$$f \leq_T g \Rightarrow f \text{ is computable by an Turing machine with advice } \alpha^g.$$

Infinite operation

Another resource which is used by some models from the literature to gain hypercomputational power is infinite operation. Models that use the resource of infinite operation are allowed to perform an infinite number of computational steps.

There seem to be two kinds of models using infinite operation. On the one hand, there are models that compute on a finite input string and produce a finite result after finitely or infinitely many computational steps. The underlying model of computation remains a standard Turing machine. Like Turing machines, these models compute values of functions from Σ^* to Σ^*, or solve decision problems. Examples of such models include accelerated Turing machines [16] and infinite time Turing machines [25]. The accelerated Turing machine performs infinitely many computational steps in finite time. It performs each step in half the time used for the step before. The first step is performed in 1 time unit, the second step in $\frac{1}{2}$ time unit, the third one in $\frac{1}{4}$ time unit, etc. The total time needed for an infinity of computational steps is $1+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\ldots < 2$ time units. To illustrate the hypercomputational power of accelerated Turing machines, we explain how the halting function can be computed by an accelerated Turing machine. This example is based on [42].

Example 3.2.1. Let A be an accelerated universal Turing machine. A starts with a 0 in the first square of its output tape. On input $\langle e, x\rangle$, A simulates the e-th Turing machine T on input x. If T halts on input x, A outputs a 1. Otherwise, A leaves the symbol in its first square a 0. The result of A's computation is considered to be the symbol in the first square of its tape after 2 time units.

On the other hand, there are models which are inspired by a different view of computing, reflecting that of contemporary information processing systems, such as operating systems [70]. Instead of computing finite function values on finite inputs, using infinitely many computational steps, these models process infinite streams of symbols or streams of finite symbol packages in an ongoing fashion. In the computations they carry out, there is no notion of a final result, as the models are not constructed to halt. A theory of ω-automata — automata that compute on infinite inputs — can be found in [55].

Although adding the ability to perform computations on infinite inputs does not change the internal behaviour of the models, it does allow the possibility for the models to receive non-Turing-computable inputs. These inputs can be translated into non-Turing-computable outputs in an effective way, for example by just computing the characteristic function of the input.

Interaction

Closely related to infinite operation is the resource of interaction. The need for a theory of interactive computation was motivated by a paradigm shift from sequential computing towards concurrent computing. In his Turing award lecture, Milner [39, p. 80] writes of his conviction that "a theory of concurrency and interaction requires a new conceptual framework, not just a refinement of what we find natural for sequential computing". In his seminal paper "Why interaction is more powerful than algorithms", Wegner [66] argues that theoretical computer science should make a paradigm shift from rule-based algorithms towards interaction, paralleling the paradigm shift from logic-based to object-based or agent-oriented models in software engineering. He argues that "[t]hough object-based programming has become a dominant practical technology, its conceptual framework and theoretical foundations are still unsatisfactory" [66, p. 82]. Wegner proposes a new class of machines as a basis for the interactive paradigm: interaction machines. He claims that the behaviour of interaction machines is not reducible to Turing machine behaviour. Therefore, he argues, interaction is more powerful than algorithms.

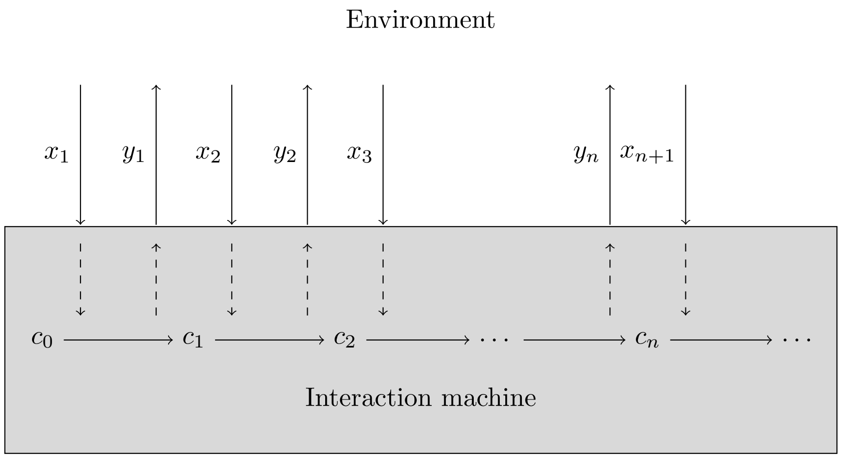


Figure 3.1: Interaction.

A Turing machine, Wegner points out, is provided with a finite input prior to the computation, but cannot accept external input during computation. This makes the Turing machine a closed system. However, Wegner argues, tasks like driving home from work can not be realised by closed, non-interactive systems, that do not take notice of interactive events in the external environment while computing. He proposes the interaction machine: a standard Turing machine extended with the ability to dynamically interact with the external environment, which it cannot control. Thereto it has input and output actions to its disposal. These input and output actions turn interaction machines into open systems. Input and output actions can either be synchronous or asynchronous, and moreover differ along other axes. Wegner [66, p. 83] states that "[i]nteraction machines can model objects, software engineering applications, robots, intelligent agents, distributed systems, and networks, like the Internet and the World-Wide-Web". Objects, for example, interact with the environment through an interface, and their outputs depend on changes of states controlled by unpredictable external actions.

Wegner points out that whereas the behaviour of Turing machines is defined by Turing-computable functions on finite input strings, the behaviour of interaction machines is defined on interaction histories. Therefore, he argues, interactive systems are able to learn from experience and adapt to the environment. However, Wegner [66] does not make explicit what he means by interaction histories, nor does he give a formal definition of interaction machines and interaction. Still, Wegner [66, p. 83] claims that "*[i]nteraction-machine behavior is not reducible to Turing-machine behavior*". As formal evidence of this irreducibility, Wegner [66,

p. 83] gives that "*[i]nput streams of interaction machines are not expressible by finite tapes, since any finite representation can be dynamically extended by uncontrollable adversaries*". He also challenges the Church-Turing thesis when he states that "when the intuitive notion of what is computable is broadened to include interactive computations, Church's thesis breaks down. Though the thesis is valid in the narrow sense that Turing machines express the behavior of algorithms, the broader assertion that algorithms capture the intuitive notion of what computers compute is invalid." [66, p.81]. Note that in fact, Wegner does not challenge the Church-Turing thesis in the narrow sense that all algorithmic computation is captured by Turing machines, but rather what Goldin and Wegner [24] call a misinterpretation of the Church-Turing thesis.

Wegner's [66] claim about the hypercomputational power of interaction machines is subject to criticism. Prasse and Rittgen [45] acknowledge Wegner's achievements in recognising the importance of investigating the influence of interaction on the performance of computers. However, they point out that since Wegner [66] does not formalise his interaction machine, the validity of his claims concerning the power of interaction can hardly be assessed. Likewise, Van Leeuwen [59] points out that, although Wegner's [66] article was very readable, as well as stimulating, the claim that interaction is more powerful than algorithms did not seem to be substantiated well enough.

Prasse and Rittgen stress that the internal behaviour of interaction machines does not differ from that of equivalent Turing machines. Therefore, they argue, an interaction machine *itself* does not possess greater computational power than standard Turing machines. However, the possibility to communicate allows the machine to utilise computational capabilities of other machines. If this is indeed what Wegner means by interaction, Prasse and Rittgen explain, interaction can be compared to a subroutine call. Then, interaction machines are models of relative computation, comparable to o-machines. Prasse and Rittgen note that, like o-machines, interaction machines have an incomplete specification, and can not compute any function by themselves. A specific oracle needs to be supplied in order for an o-machine to solve a certain problem. Likewise, an interaction machine for solving a concrete problem needs a suitable interaction partner. An increase in computational power beyond that of Turing machines relies on the external interaction partner, not on the possibility to interact itself. Prasse and Rittgen argue that interaction machines are not themselves algorithms, but rather components of interactive systems. In fact, these interactive systems are protocols, which specify how procedures are performed by the interaction machine components, and how interaction proceeds.

Prasse and Rittgen point out that it is not obvious what it is an interaction machine computes, if it actually computes anything at all. The computation depends on input from the environment, which unpredictable. The environment generally refers to those components which do not belong to the system under consideration. Prasse and Rittgen suggest that one solution might be to consider machine-environment pairs, in which case the corresponding function is from environment input streams to machine output-streams. However, in this case, Prasse and Rittgen add, the overall function is derived by integrating the environment into the computation. This adheres to closing the system. Therefore, Prasse and Rittgen argue, "the difference between open and closed systems 'lies in the eye of the beholder'" [45, p. 359].

Van Leeuwen [59] points out that Wegner seemed to have simply changed the rules of computation by allowing inputs to continuously stream into the machine, thereby allowing the inputs to be potentially infinite. According to Van Leeuwen, Wegner did not seem to realise that a model of such computations already existed in ω-Turing machines. Wegner's article motivated Van Leeuwen and Wiedermann to formalise the notion of interaction in [60]. Van Leeuwen and Wiedermann conclude that "[f]rom the viewpoint of computability theory, interactive computation e.g. with ITMs does not lead to super-Turing computing power. Interactive computing merely extends our view of classically computable functions over finite domains to computable functions (translations) defined over infinite domains. Interactive computers simply compute something different from non-interactive ones because they follow a different scenario" [70, p. 582]. Moreover, "[r]emembering the respective inputs over time, a finite computation of an interactive machine can always be replayed *a posteriori* by a non-interactive machine giving the same outputs as the interactive machine" [70, p. 582]. The following definition is based on [60].

Definition 3.2.3 (Monotonic function). *A partial function $f : 2^* \to 2^*$ is called* monotonic *if for all $x, y \in 2^*$, if $x \prec y$ (x is finite and a strict prefix of y) and $f(y)$ is defined, then $f(x)$ is defined as well and $f(x) \preceq f(y)$.*

Van Leeuwen and Wiedermann [60, 63] modify the classical definition of continuous functions to the case of functions on infinite strings as follows.

Definition 3.2.4 (Limit-continuous function). *A partial function $f : 2^\omega \to 2^\omega$ is called* limit-continuous *if there exists a Turing-computable partial function $g : 2^* \to 2^*$ such that:*

- *g is monotonic; and*
- *for all strictly increasing chains $u_1 \prec u_2 \prec \ldots \prec u_t \prec \ldots$ with $u_t \in 2^*$ for $t \geq 1$, it holds that $f(\lim_{t\to\infty} u_t) = \lim_{t\to\infty} g(u_t)$, as soon as either the left-hand side or the right-hand side of the equality is defined.*

Van Leeuwen and Wiedermann [60] point out that all interactively computable functions share the property that on finite parts of infinite strings on which they are defined they are continuous. That is, at any moment, any further extension of the input should lead to an extension of the output as it is at that moment. The following theorem is from [60, p. 110].

Theorem 3.2.2. *If $f : 2^\omega \to 2^\omega$ is interactively computable, then f is limit-continuous.*

For a proof of this theorem, we refer to [60]. In [63, p. 140], Van Leeuwen and Wiedermann conclude that "When considering only finite computations, there is no difference between the power of classical and interactive computations. Keeping the classical computation time-bounded on the one hand and considering infinite interactive computations on the other, is to draw a comparison between two incomparable things: while the former computes with finite objects (finite streams), the latter operates on infinite objects. Thus, the two modes are incomparable; each of them computes with different entities. Therefore it is not possible to say which of the two has a greater computational power. However, our results show that in the limit the computational power in both modes tends to coincide".

Here, we explore the possibility of viewing interactive Turing machines as defined in Section 2.2 as models of relative computation. This would allow us to measure the hypercomputational power of Van Leeuwen and Wiedermann's models later on in Section 3.3. The idea is to look at the stream producible by an interactive Turing machine and interaction partner pair. Given an interaction partner E, an interactive Turing machine M can only produce a single output stream. In fact, every interactive Turing machine thus computes a functional where the argument is an interaction partner and the output an infinite stream. We will show that this functional is comparable to the functional computed by o-machines producing streams from a blank input tape. The idea of Turing machine based models producing infinite streams instead of computing functions from Σ^* to Σ^* is not new, and was expressed by Turing himself in [56].

First, we redefine what it is interactive Turing machines compute. Thereto, we first define the notion of a 'suitable interaction partner'.

Definition 3.2.5 (Suitable interaction partner)**.** *Let M be an interactive Turing machine with input alphabet Σ and output alphabet Γ. Let E be an environment with input alphabet Γ and output alphabet Σ. Then E is a suitable interaction partner of M if and only if E produces a λ symbol at each time when M is in an internal state, and M and E satisfy the interactiveness condition.*

Instead of interactively realisable translations from Σ^ω to Γ^ω, we define interactive producibility on infinite output streams.

Definition 3.2.6 (Interactively producible stream)**.** *Let M be an interactive Turing machine with input alphabet Σ and output alphabet Γ. Let E be an environment. Let*

$$\Phi_M^E = \begin{cases} \textit{the stream produced by } M \textit{ in interaction with } E & \textit{if } E \textit{ is suitbale to } M \\ \textit{undefined} & \textit{otherwise} \end{cases}.$$

A stream $y \in \Gamma^\omega$ is called interactively producible if there exists an interactive Turing machine M and an environment E such that $\Phi_M^E = y$.

Now we associate with each possible environment a 'comparable oracle', defined as follows.

Definition 3.2.7 (Comparable oracle)**.** *Let E be an environment, let Γ denote its input alphabet, and Σ its output alphabet. A comparable oracle has the same input and output alphabet as E. Let x_1 denote the first input symbol sent by E to the interaction machine. Construct Θ_E such that on query "ε?" it answers with x_1. For each $n \in \mathbb{N}$, enumerate all words over Γ of length n. Let $y_{n,1}, y_{n,2}, ..., y_{n,|\Gamma|^n}$ be such an enumeration. Let $x_{n,1}, x_{n,2}, ..., x_{n,|\Gamma|^n}$ denote E's reactions to the respective combinations of first n output symbols of the interaction machine, with each $x_{n,i} \in \Sigma \cup \{\lambda\}$. Now construct Θ_E such that to each query of the form "$\Theta(y_{n,i})$" it responds with $x_{n,i}$ if $x_{n,i} \in \Sigma$, and with ε if $x_{n,i} = \lambda$, for all $n \in \mathbb{N}$.*

We also define a comparable o-machine for every interactive Turing machine. A comparable o-machine is an o-machine without an input tape, but with a work tape, an output tape, and an oracle tape. The output tape also serves as query tape.

Definition 3.2.8 (Comparable o-machine). *Let $M = \langle Q, \Sigma, \Gamma, I, q_0, \delta \rangle$ be an interactive Turing machine. Without loss of generality, we assume that M has a single input port and a single output port, as well as a single work tape. Then we construct a comparable o-machine $M' = \langle Q', \Sigma, \Gamma, \Omega, A, q_0, \delta' \rangle$ as follows.*

- *$A = \{(q, query) \mid q \in (Q - I)\}$ is the set of query states;*
- *$Q' = A \cup \{q \mid q \in I\}$ is the set of states;*
- *$\Omega = \Sigma$ is the oracle alphabet;*
- *$q_0 \in A$ is the initial state;*
- *δ' is the transition function such that for each transition tuple*

  $$\delta(\langle q, w, i \rangle, \sigma) = \langle \langle r, w', i' \rangle, \gamma \rangle$$

  of M, M' has the transition tuple

  $$\delta'(\langle q', w, i, y, k, o, 1 \rangle) = \langle r', w', i', y', k', o', 1 \rangle$$

  satisfying the following conditions.

  - *if q is internal, then $q' = q$ is not a query state, and $o = \varepsilon$;*
  - *if q is external, then $q' = (q, query)$ is a query state, and $o = \Theta(y)$;*
  - *if r is internal, then $r' = r$ is not a query state, $y' = y$, $k' = k$, and $o' = \varepsilon$;*
  - *if r is external, then $r' = (r, query)$ is a query state, $y' = y\gamma$, and $k' = k + 1$.*

A comparable o-machine M' with oracle Θ computes as follows (see also Figure **??**). At time $t = 0$, before the start of the computation, M is in its initial state q_0, which is a query state. In the first step, M' queries Θ for the value of the empty string ε, because the output/query tape is then empty. The oracle returns $\Theta(\varepsilon) \in \Omega$. This symbol can be interpreted as the first input symbol to M'. Then, M' makes its transition based on its current configuration and the oracle value, and either appends a symbol $y_1 \in \Gamma$ or the empty string ε to the contents of its output tape, depending on whether the new state is a query state or not. In the following step, there are two cases. First, if M' is in a query state, M' queries its oracle. The string on its query tape is then y_1. The oracle responds with $\Theta(y_1) \in \Omega$, and M' makes a transition according to its current configuration and the oracle value. Otherwise, if M' is in a state which is not a query state, it doesn't query the oracle during this step, and makes a transition based on its current configuration and a blank oracle symbol. In general, in each time step t, if M' is in a query state, M' queries its oracle for the value $\Theta(y_1 y_2 \ldots y_m)$, where $y_1 y_2 \ldots y_m$ denotes the content of M''s output/query tape, and makes a transition. If M' is in a state which is not a query state, it doesn't query its oracle, and makes a transition. If the new state is a query state, M' appends a symbol $y_t \in \Gamma$ to the contents of its output/query tape, and moves the tape head one step right. Otherwise, the contents of the output/query tape remain unchanged. The stream $x = \Theta(\varepsilon)\Theta(y_1)\Theta(y_1 y_2)\Theta(y_1 y_2 y_3) \ldots$ can be interpreted as input to M', and $y = y_1 y_2 y_3 \ldots$ as M's output. We denote the output stream produced by M' with oracle Θ in this way by $\Phi^{\Theta}_{M'}$.

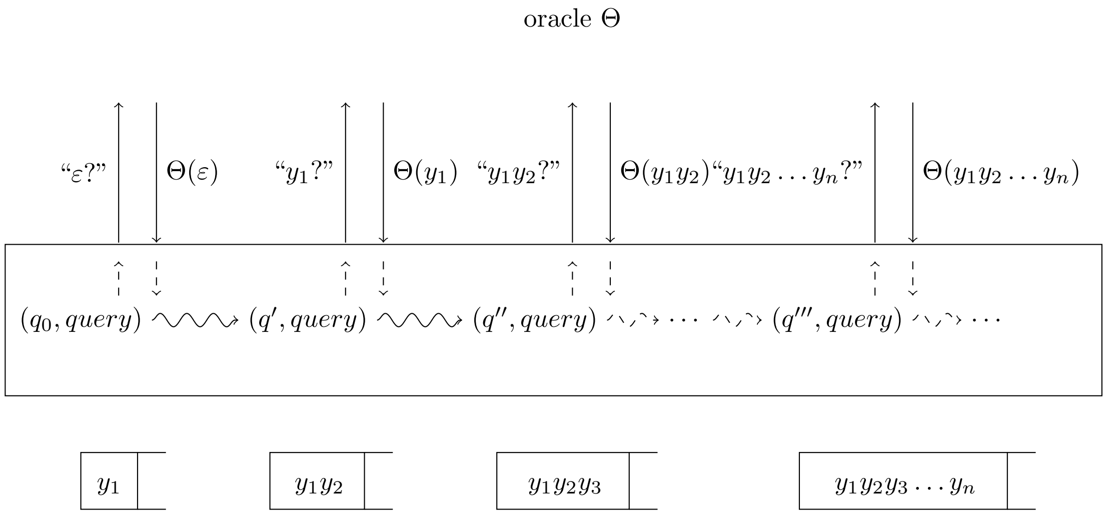


Figure 3.2: Stream production by o-machine M' with oracle Θ.

Now our aim is to show that for every interactively producible stream y, if y is produced by M in interaction with environment E, it can also be produced by comparable o-machine M' with comparable oracle Θ^E. Then, in a sense it can be said that stream y is Turing-reducible to Θ^E. Therefore, we argue, we can measure that power of interactive Turing machines in terms of the streams they can produce relative to the degree of unsolvability of their environment.

Lemma 3.2.3. *Let Φ_M^E be the stream producible by an interactive Turing machine M in interaction with environment E. Let $\Phi_{M'}^{\Theta^E}$ be the stream producible by comparable o-machine M' with comparable oracle Θ. Then $\Phi_{M'}^{\Theta^E} = \Phi_M^E$.*

Proof. We show that at each time the sequence of oracle values provided by Θ^E to M' up to that time is the filtered version of the input stream provided by the environment E to interactive Turing machine M up to that time. Moreover, we show that at each time the output on M''s output tape is equal to the filtered version of the output produced by M up to that time.

Suppose at time t, interactive Turing machine M is in state q. Suppose M has received n non-λ input symbols up to now, together forming the sequence filtered$(x_1x_2...x_t) = f_1f_2...f_n$. Moreover, suppose M has produced m non-λ outputs symbols up to now, which together form the sequence filtered$(y_1y_2...y_t) = u_1u_2...u_m$. At this time, M' has queried its oracle n times, and has received the sequence $f_1f_2...f_n$ of oracle answers. Moreover, M' has $u_1u_2...u_m$ on its output tape. There are four cases to consider.

- First, consider the case that q is an internal state. Then M' is in state q as well, which is not a query state. At time $t+1$, M receives the next input symbol x_{t+1} from E. Since q is an internal state, it is required that $x_{t+1} = \lambda$. Now M makes a transition according to $\delta(\langle q, w, i\rangle, \lambda) = \langle\langle q', w', i'\rangle, \gamma\rangle$. Again, there are two cases to consider.

– First, if q' is internal, $\gamma = \lambda$. Hence, M makes a transition according to

$$\delta(\langle q, w, i\rangle, \lambda) = \langle\langle q', w', i'\rangle, \lambda\rangle.$$

Now M' makes a transition according to

$$\delta'(\langle q, w, i, u_1u_2...u_m, m+1, \varepsilon, 1\rangle) = \langle q', w', i', u_1u_2...u_m, m+1, \varepsilon, 1\rangle.$$

After this transition, filtered$(y_1y_2...y_ty_{t+1})$ = filtered$(y_1y_2...y_t) = u_1u_2...u_m$, because $\gamma = \lambda$. Likewise, after this transition M''s output tape still contains $u_1u_2...u_m$ (see Figure 3.3(a) and Figure 3.3(b)).

– Second, if q' is external, $\gamma \in \Gamma$. Hence, M makes a transition according to

$$\delta(\langle q, w, i\rangle, \lambda) = \langle\langle q', w', i'\rangle, \gamma\rangle.$$

Now M' makes a transition according to

$$\delta'(\langle q, w, i, u_1u_2...u_m, m+1, \varepsilon, 1\rangle) = \langle (q', query), w', i', u_1u_2...u_m\gamma, m+2, \varepsilon, 1\rangle.$$

After this transition, filtered$(y_1y_2...y_ty_{t+1})$ = filtered$(y_1y_2...y_t) = u_1u_2...u_m\gamma$, because $\gamma \in \Gamma$. Likewise, after this transition M''s output tape contains $u_1u_2...u_m\gamma$ (see Figure 3.3(c) and Figure 3.3(d)).

In both cases, after this transition filtered$(x_1x_2...x_tx_{t+1})$ = filtered$(x_1x_2...x_t) = f_1f_2...f_n$, because $x_{t+1} = \lambda$. Likewise, after this transition M' has not queried its oracle, and not received an oracle answer. Therefore, the sequence of oracle answers received up to now is still $f_1f_2...f_n$.

- Second, consider the case that q is an external state. Then M' is in query state $(q, query)$. At time $t+1$, M receives the next input symbol x_{t+1} from E. This symbol can be any symbol in $\sigma \in \Sigma \cup \{\lambda\}$. Now M makes a transition according to

$$\delta(\langle q, w, i\rangle, \sigma) = \langle\langle q', w', i'\rangle, \gamma\rangle.$$

Now M' is in a query state, and therefore first queries its oracle for the value $\Theta(u_1u_2...u_m)$. Since Θ is comparable to E, the value $\Theta(u_1u_2...u_m)$ will be equal to σ if $\sigma \in \Sigma$ and equal to ε if $\sigma = \lambda$. Again, there are two cases to consider.

– First, if q' is internal, $\gamma = \lambda$. Hence, M makes a transition according to

$$\delta(\langle q, w, i\rangle, \sigma) = \langle\langle q', w', i'\rangle, \lambda\rangle.$$

Now M' makes a transition according to

$$\delta'(\langle (q, query), w, i, u_1u_2...u_m, m+1, \sigma, 1\rangle) = \langle q', w', i', u_1u_2...u_m, m+1, \varepsilon, 1\rangle.$$

After this transition, filtered$(y_1y_2...y_ty_{t+1})$ = filtered$(y_1y_2...y_t) = u_1u_2...u_m$, because $\gamma = \lambda$. Likewise, after this transition M''s output tape still contains $u_1u_2...u_m$ (see Figure 3.4(a) and Figure 3.4(b)).

– Second, if q' is external, $\gamma \in \Gamma$. Hence, M makes a transition according to

$$\delta(\langle q, w, i\rangle, \sigma) = \langle\langle q', w', i'\rangle, \gamma\rangle.$$

Now M' makes a transition according to

$$\delta'(\langle(q, query), w, i, u_1u_2...u_m, m{+}1, \sigma, 1\rangle) = \langle(q', query), w', i', u_1u_2...u_m\gamma, m{+}2, \varepsilon, 1\rangle.$$

After this transition, filtered$(y_1y_2...y_ty_{t+1})$ = filtered$(y_1y_2...y_t) = u_1u_2...u_m\gamma$, because $\gamma \in \Gamma$. Likewise, after this transition M''s output tape contains $u_1u_2...u_m\gamma$ (see Figure 3.4(c) and Figure 3.4(d)).

In both cases, after this transition

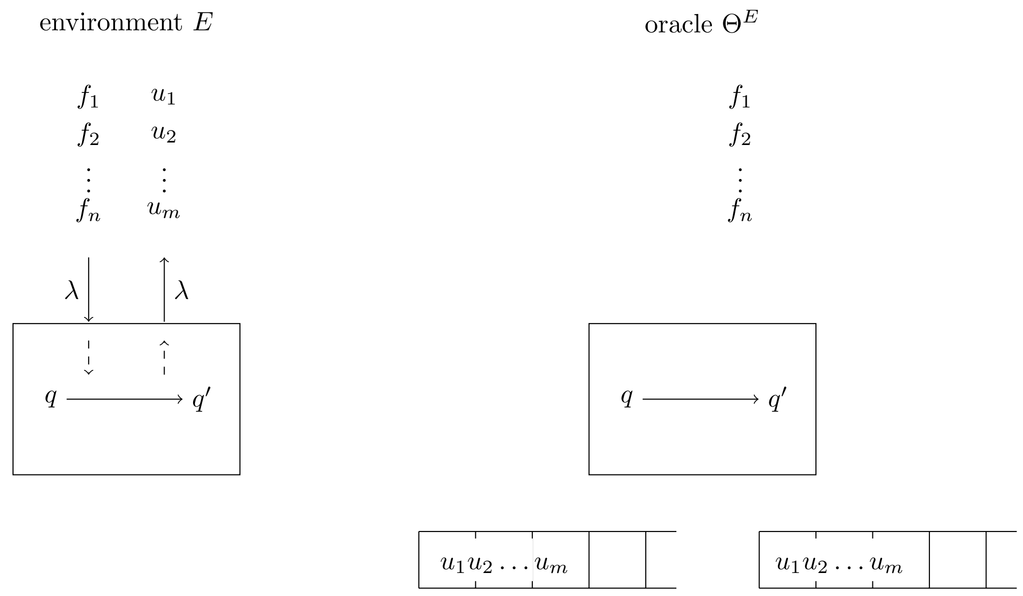
$$\text{filtered}(x_1x_2...x_tx_{t+1}) = \begin{cases} \text{filtered}(x_1x_2...x_t) = f_1f_2...f_n & \text{if } x_{t+1} = \lambda \\ \text{filtered}(x_1x_2...x_tx_{t+1}) = f_1f_2...f_nx_{t+1} & \text{if } x_{t+1} \in \Sigma. \end{cases}$$

Likewise, after this transition M' has queried its oracle, and the sequence of oracle answers received up to now is

$$f_1f_2...f_n\Theta(u_1u_2...u_m) = \begin{cases} f_1f_2...f_n\varepsilon = f_1f_2...f_n & \text{if } x_{t+1} = \lambda \\ f_1f_2...f_nx_{t+1} & \text{if } x_{t+1} \in \Sigma. \end{cases}$$

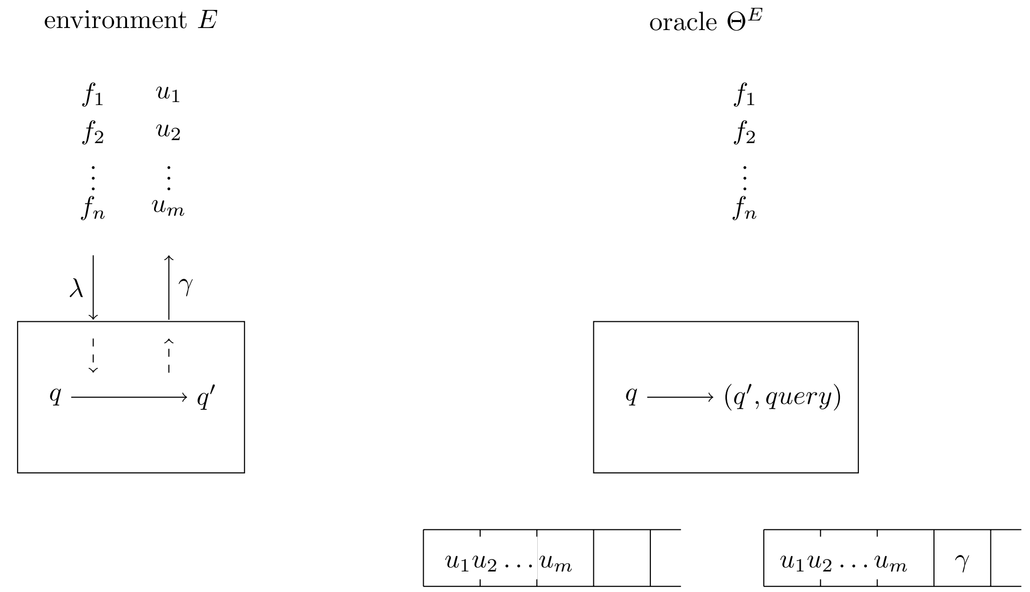
□

Now we have that if $y \in \Gamma^\omega$ is interactively producible with respect to environment E, then y is producible by an o-machine with comparable oracle Θ^E. Hence, we conjecture than if y is interactively producible with respect to environment E, then $y \leq_T \Theta^E$.



(a) In case M makes a transition to an internal state q', and outputs a λ.

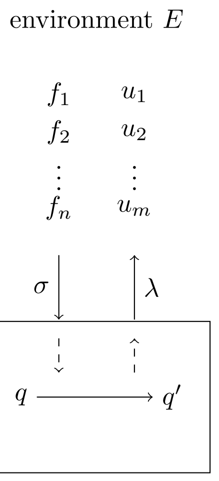
(b) M' makes a transition to q', which is not a query state, and the contents of M''s output tape remain unchanged.

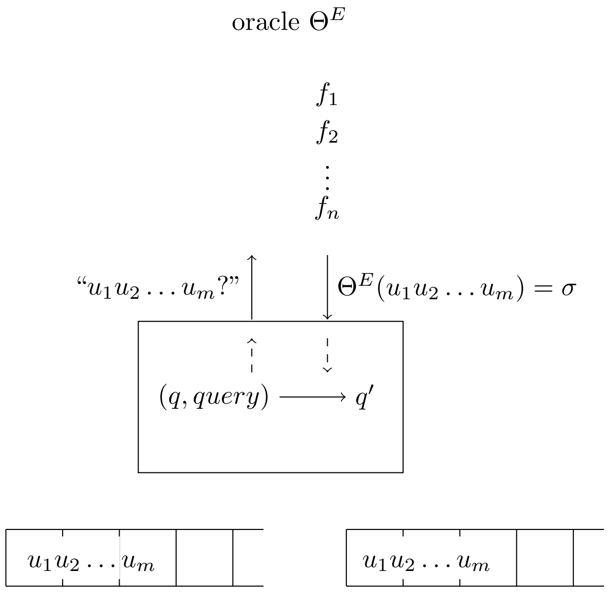


(c) In case M makes a transition to an external state q', and outputs γ.

(d) M' makes a transition to query state $(q', query)$, and appends γ to the end of the contents of its output tape.

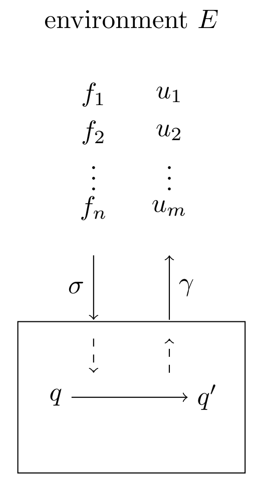
Figure 3.3: M starts in an internal state, receives a λ from the environment and makes a transition. O-machine M' does not query its oracle during the simulation of M's this transition.

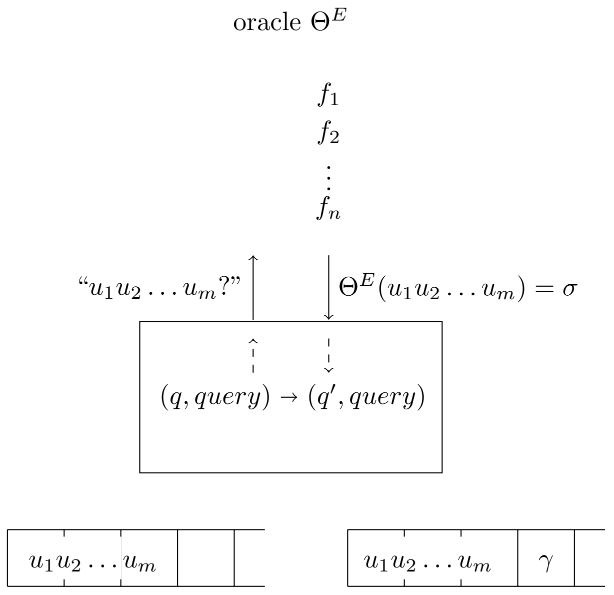




(a) In case M makes a transition to an internal state q', and outputs a λ.

(b) M' makes a transition to q', which is not a query state, and the contents of M''s output tape remain unchanged.





(c) In case M makes a transition to an external state q', and outputs γ.

(d) M' makes a transition to query state $(q', query)$, and appends γ to the end of the contents of its output tape.

Figure 3.4: M starts in an external state, receives γ from the environment and makes a transition. O-machine M' queries its oracle during the simulation of M's this transition.

Evolution

Another resource which is relevant to Van Leeuwen and Wiedermann's [68, 69] models is the resource of evolution. Van Leeuwen and Wiedermann's [69] lineages of cognitive transducers, which we introduced in Section 2.1, can be interpreted as a models of subsequent generations of cognitive transducers. On the other hand, a lineage of cognitive transducers can also be interpreted as a model of an the adaptation of an individual cognitive transducer, unfolded in time. In general, a lineage can be constructed only a posteriori, after observing the actual evolutionary process the generations of cognitive transducers, or the individual cognitive transducer have gone through respectively. Yet, for our purpose, it is useful to define an evolution function corresponding to a lineage of cognitive transducers.

Definition 3.2.9 (Evolution function). *Let $\mathcal{A} = \{A_1, A_2, ...\}$ be a lineage of cognitive transducers and let $\mathcal{M}$ denote the set of all cognitive transducers. Then the evolution function ε corresponding to $\mathcal{A}$ is defined as $\varepsilon : \mathbb{N} \to \mathcal{M}$, where*

$$\varepsilon(i) = A_i, \text{ for all } i \in \mathbb{N}.$$

Theorem 2.3.1 in Chapter 2 states that lineages of cognitive transducers are as powerful as interactive Turing machines with advice. However, the hypercomputational power of a lineage of cognitive transducers depends on the non-Turing-computability of its evolution function. Every lineage of cognitive transducers corresponding to a Turing-computable evolution function can be simulated a posteriori by an interactive Turing machine without advice. Informally, at each time, this ITM simulates the controlling cognitive transducer in the lineage until this cognitive transducer reaches a global state. Then the ITM computes the description of the next cognitive transducer in the lineage according to the Turing-computable evolution function and proceeds by simulating this cognitive transducer from the global state on the next input symbol.

Besides on the a non-Turing-computable evolution function, the hypercomputational power of a lineage of cognitive transducers depends on infinite operation. In finite time, only a finite prefix of a lineage of cognitive transducers participates in the computation. This computation can be simulated a posteriori by a single cognitive transducer.

Theorem 3.2.3. *For each lineage of cognitive transducers $\mathcal{A}$, each finite prefix of $\mathcal{A}$ can be simulated by a cognitive transducer.*

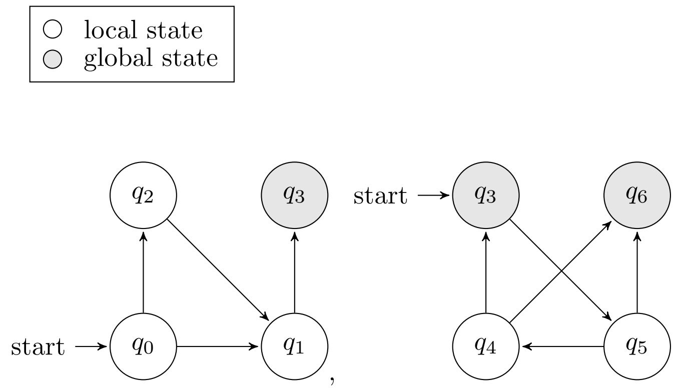
Proof. We make use of the technique of merging successive automata in a lineage outlined by Verbaan [64]. Let $\mathcal{A} = \{A_1, A_2, ...\}$ be a lineage of cognitive transducers. Denote by Q_i and $G_i \subset Q_i$ the set of states and the set of global states of $A_i \in \mathcal{A}$ respectively, for all $i \in \mathbb{N}$. Let d_i denote the transition function of $A_i \in \mathcal{A}$, for all $i \in \mathbb{N}$. Consider a length m prefix $\text{prefix}_m(\mathcal{A}) = \{A_1, ..., A_m\}$ of $\mathcal{A}$. Construct a cognitive transducer $M = \langle Q, \Sigma, \Gamma, q_0, \delta \rangle$, such that Q is the disjoint union of $Q_1, ..., Q_m$. That is

$$Q = \bigcup_{i=1...m} \{(q, i) \mid q \in Q_i\}$$

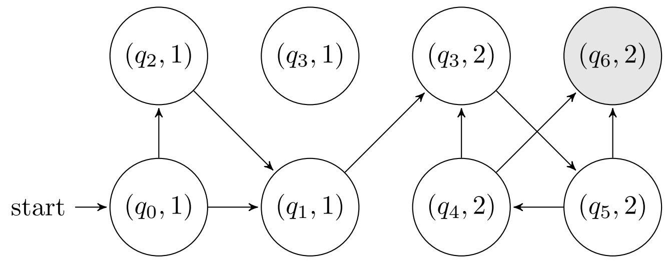
Moreover, $q_0 = (q_0, 1)$. Construct δ such that for $a \in \Sigma$ and $b \in \Gamma$, the transition $\delta((q, i), a)$ is defined

$$\delta((q,i),a) = \begin{cases} ((r,i),b) & \text{if } \delta_i(q,a) = (r,b) \text{ and } r \in Q_i - G_i \\ ((r,i+1),b) & \text{if } \delta_i(q,a) = (r,b) \text{ and } r \in G_i \\ \text{undefined} & \text{if } \delta_i(q,a) \text{ is undefined} \end{cases}$$

Since we are dealing with a finite prefix, we define a set of halting states $F_{\mathcal{A},m}$ for $\text{prefix}_m(\mathcal{A}$, such that $F_{\mathcal{A}} = G_m$. Likewise, we define a set of halting states F for M, such that $F = \{(q, m) \mid q \in G_m\}$. An example of the construction of a cognitive transducer for simulating a prefix is given in Figure 3.5.



(a) Length 2 prefix of a lineage of cognitive transducers: $\{A_1, A_2\}$.



(b) Cognitive transducer M simulating the prefix $\{A_1, A_2\}$.

Figure 3.5: A cognitive transducer constructed for simulating a prefix of a lineage.

On input $x \in \Sigma^* \cup \Sigma^\omega$, cognitive transducer M simulates $\text{prefix}_m(\mathcal{A})$ as follows. Suppose that after processing the n-th input symbol, A_k is the controlling cognitive transducer in $\text{prefix}_m(\mathcal{A})$. Suppose A_k is in state $q \in Q_k$. At this time M is in state (q, k). If $k = m$ and $q \in G_m$, A_k is in a halting state, and so is M. Otherwise, on receiving the $(n+1)$-th input symbol x_{n+1}, A_k makes a transition according to $d_k(q, x_{n+1}) = (r, y_{n+1})$ to state r, outputting y_{n+1}. There are two cases.

- If $r \in Q_k - G_k$, then A_k remains the controlling cognitive transducer. Now M's transition function δ is such that $\delta((q, k), x_{n+1}) = ((r, k), y_{n+1})$. Hence, M makes a transition to state (r, k), and outputs symbol y_{n+1}.
- Otherwise, if $r \in G_k$, A_k passes over control to A_{k+1}. Now M's transition function δ is such that $\delta((q, k), x_{n+1}) = ((r, k+1), y_{n+1})$. Hence, M makes a transition to state $(r, k+1)$, and outputs symbol y_{n+1}.

At each step, M outputs the same symbol as the prefix of $\mathcal{A}$. Cognitive transducer M realises the same transduction as $\text{prefix}_m(\mathcal{A})$. □

On the other hand, if $\text{prefix}_m(\mathcal{A})$ is simulated by cognitive transducer M, in general M can not simulate $\text{prefix}_{m+1}(\mathcal{A})$. Although a cognitive transducer M' which can $\text{prefix}_{m+1}(\mathcal{A})$ does exist, in general it is not Turing-computable from M.

Asynchrony

Asynchrony is another resource which can be used to gain hypercomputational power. Copeland and Sylvan [19] introduce asynchronous networks of Turing machines as a hypercomputational model. Copeland and Sylvan illustrate their claim that asynchronous networks of Turing machines can invoke hypercomputation by an example, in which two Turing machines, M_1 and M_2 operate on a shared tape. M_0 only prints 0's and M_1 only prints 1's. Both Turing machines only write on blank portions of the tape, and only on those cells whose predecessors have already been written on. With each Turing machine M_i is associated a timing function Δ_i, where $\Delta_i(n) = k$ if and only if k moments of operating time separate the n^{th} atomic operation of M_i from its $n + 1^{th}$ operation. Copeland and Sylvan state that if M_0 and M_1 operate asynchronously, and at least one of Δ_0 and Δ_1 is not Turing-computable, then M_0 and M_1 are possibly in the process of inscribing an non-Turing-computable real number on the tape.

Actually, if M_0 and M_1 alternate their printing randomly, they can be shown to inscribe a non-Turing-computable infinite string on the tape almost surely (i.e. with probability one).

Proof. We use a measure theoretical argument. Let I denote the set of all right-infinite strings over $\{0, 1\}$. This set I is uncountable. Let E denote the subset of all Turing-computable right-infinite strings over $\{0, 1\}$. This set E, on the other hand, is countable. There is a theorem in measure theory that states that countable subsets have measure zero, so $\mu(E) = 0$. The complement of E therefore has full measure, so $\mu(I - E) = 1$. Hence, the probability that infinite string produced is non-Turing-computable is 1. □

On the other hand, when both Δ_0 and Δ_1 are Turing-computable, or M_0 and M_1 operate synchronously, the resulting network of Turing machines is equivalent to a single Turing machine.

Infinite specification

Finally, we discuss the resource of infinite specification . In [42], Ord discusses the infinite state Turing machine. An infinite state Turing machine is a Turing machine whose set of states is allowed to be infinite. More specifically, an infinite state Turing machine can have a state for every input string over its alphabet Σ. In this way, for each function from $\mathbb{N}$ to $\{0, 1\}$, including for example the halting function, or for each function from Σ^* to Σ^*, an infinite state Turing machine can be specified. Figure 3.2 shows a template of an infinite state Turing machine with alphabet $\Sigma = \Gamma = \{0, 1\}$. The infinite state Turing machine has a special state q_w for every possible finite input string w over its input alphabet. It also has a special state p_v for every possible finite output string v over its output alphabet. The transition arrows between q_i and q_j are labelled with operations on the machine's input

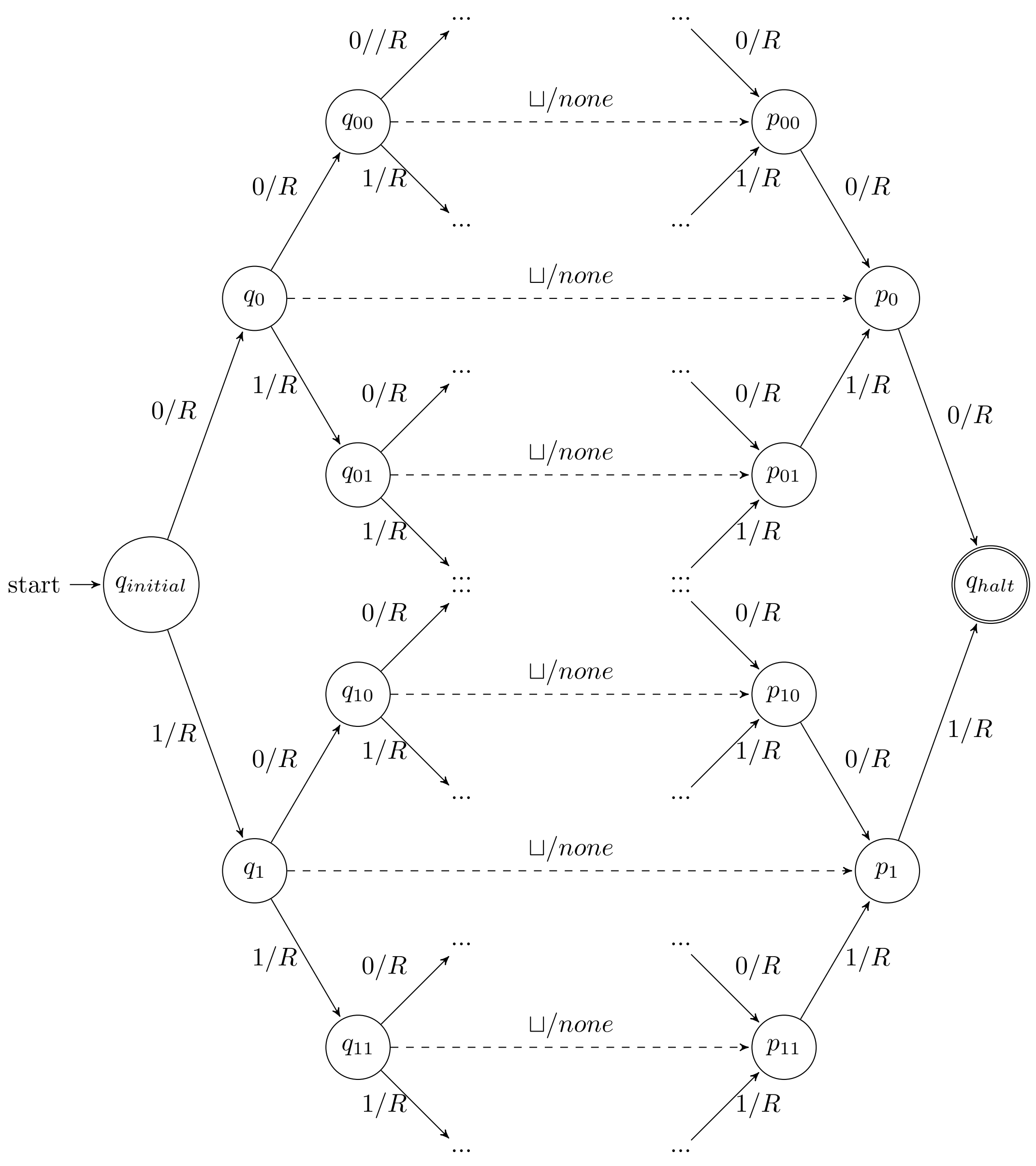


Figure 3.6: Template for an infinite state Turing machine with alphabet $\{0, 1\}$.

tape, for all $i, j \in \Sigma^*$. The transition arrows between p_k and p_l are labelled with operations on the machine's output tape, for all $k, l \in \Sigma^*$. By rearranging the dotted transition arrows, any function from Σ^* to Γ^* can be specified.

3.3 Hypercomputational power

Hypercomputational power is the ability to compute functions, or solve problems that cannot be solved by standard Turing machines. In this section, we take a more detailed look into the world beyond the Turing-computable. It turns out that this world is hierarchically structured. This structure allows for a more specific characterisation of the hypercomputational power of hypercomputational models. We introduce the arithmetical hierarchy and use it as a framework for measuring the relative powers of hypercomputational models.

The arithmetical hierarchy

The arithmetical hierarchy is a classification of sets based on the complexity of the formulas in first-order arithmetic that define them. The first level of the arithmetical hierarchy is defined as follows [12, p. 74].

Definition 3.3.1 (Level 1 of the Arithmetical Hierarchy). *1. If $\forall x \in \mathbb{N}$ we have $x \in A \iff (\exists y)R(x,y)$ for some computable relation R then we say that A is a Σ^0_1 set, and write $A \in \Sigma^0_1$.*

2. If we have $x \in A \iff \forall y[R(x,y)]$, some computable R, then we say that A is a Π^0_1 set, and write $A \in \Pi^0_1$.

3. If $A \in \Sigma^0_1 \cap \Pi^0_1$, then we say that A is a Δ^0_1 set, and write $A \in \Delta^0_1$.

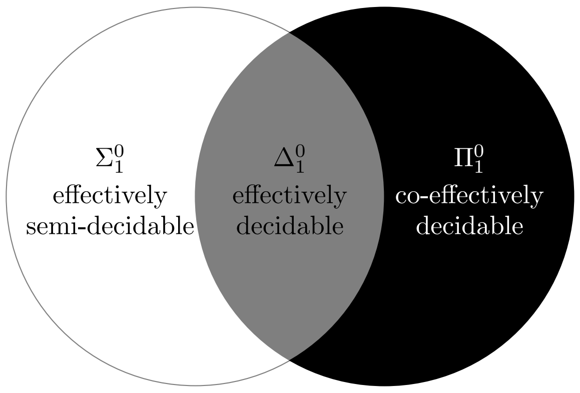


Figure 3.7: The first level of the arithmetical hierarchy.

The classes of the first level of the arithmetical hierarchy are related to the effectively decidable, semi-decidable, and co-semi-decidable sets in the following way.

Theorem 3.3.1. *Σ^0_1 coincides with the set of effectively enumerable, or effectively semi-decidable sets.*

Proof. This proof is based on [12].

($\Rightarrow$) Suppose A is effectively enumerable. Then, by Definition 3.1.10, $A = \emptyset$ or $A = \text{range}(f)$ for some total Turing-computable function f. If $A = \emptyset$ then

$$x \in A \iff \exists s(x + 1 = x).$$

Since $(x + 1) = x$ is a Turing-computable relation of x and s, by Definition 3.3.1 $A \in \Sigma^0_1$. Otherwise, if $A = \text{range}(f)$ for some total Turing-computable function f,

$$x \in A \iff \exists s(f(s) = x).$$

Since $f(s) = x$ is a Turing-computable relation of x and s, $A \in \Sigma^0_1$.

($\Leftarrow$) Suppose $A \in \Sigma^0_1$. Then, $x \in A \iff \exists s(R(x, s)$ for some Turing-computable relation R. Consider the following function:

$$\psi(x) = \begin{cases} 1 & \text{if } \exists s(R(x,s)) \\ \text{undefined} & \text{otherwise} \end{cases} .$$

The function ψ is Turing-computable; given x, we can subsequently check whether $R(x, 1), R(x, 2), ...$ holds, until we come across an s for which $R(x, s)$ holds. If this search never terminates, we have $\psi(x)$ is undefined as required. Hence, there is an e such that $\phi_e = \psi$. Now

$$x \in A \iff \psi(x) \text{ is defined.}$$

Hence, $A = \text{dom}(\psi)$, the domain of a Turing-computable function. Therefore, by definition, A is effectively enumerable.

□

Theorem 3.3.2. Π^0_1 *coincides with the set of co-effectively enumerable sets.*

Theorem 3.3.3. Δ^0_1 *coincides with the set of effectively decidable sets.*

For a proof of Theorem 3.3.2 and Theorem 3.3.3 we refer to a standard textbook on computability theory, such as [12]. The halting set K_0 is an example of a set in Σ^0_1, since it is effectively enumerable, but not effectively decidable.

The first level of the arithmetical hierarchy can be extended to higher levels. The following definition is from [12, p. 154]

Definition 3.3.2 (The Arithmetical Hierarchy). *1.* $\Sigma^0_0 = \Pi^0_0 = \Delta^0_0 =$ *all computable relations. And for* $n \geq 0$*:*

2. $\Sigma^0_{n+1} =$ *all relations of the form* $(\exists \vec{y_l})R(\vec{x}_k, \vec{y_l})$*, with* $R \in \Pi^0_n$.

3. $\Pi^0_{n+1} =$ *all relations of the form* $(\forall \vec{y_l})R(\vec{x}_k, \vec{y_l})$*, with* $R \in \Sigma^0_n$.

4. $\Delta^0_{n+1} = \Sigma^0_{n+1} \cap \Pi^0_{n+1}$.

R *is* **arithmetical** *if* $R \in \bigcup_{n \geq 0}(\Sigma^0_n \cup \Pi^0_n)$.

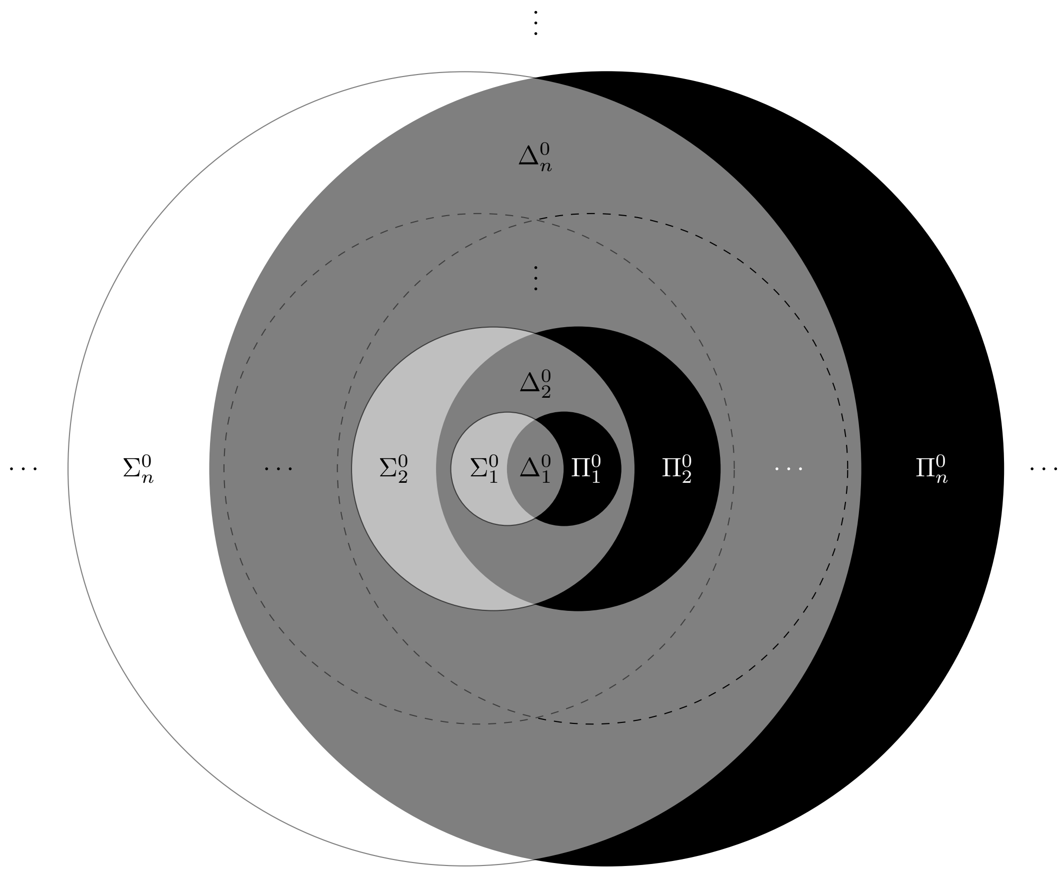


Figure 3.8: The arithmetical hierarchy.

It holds that

$$\Delta_0^0, \Sigma_0^0, \Pi_0^0 \subseteq \Delta_1^0 \subseteq \Sigma_1^0, \Pi_1^0 \subseteq ... \subseteq \Sigma_n^0, \Pi_n^0 \subseteq \Delta_{n+1}^0 \subseteq \Sigma_{n+1}, \Pi_{n+1} \subseteq ...$$

Since we can identify functions with their graphs, we can say that a function f is classified into the class in the arithmetical hierarchy to which its graph belongs. In other words, a function $f \in \Sigma_n^0 \iff \text{Graph}(f) \in \Sigma_n^0$, and $f \in \Pi_n^0 \iff \text{Graph}(f) \in \Pi_n^0$. By Proposition 3.1.1 and Proposition 3.1.2, Σ_1^0 then contains the partial Turing-computable functions, and Δ_1^0 contains the total Turing-computable functions.

The arithmetical hierarchy captures all relations describable in true first order arithmetic — that is, relations which are definable in Peano arithmetic. Cooper [14, p. 1355] points to the difference in definability on the one hand, and computability on the other hand, for which the arithmetical hierarchy is a metaphor: “[w]hat the arithmetical hierarchy encapsulates is the smallness of the computable world in relation to what we can describe”. Since only the sets in Δ_1^0 are effectively decidable, any model or machine deciding a set outside Δ_1^0 is capable of hypercomputation. The arithmetical hierarchy also shows that the world beyond the Turing-computable is not a total chaos, but is structured indeed.

The usefulness of the arithmetical hierarchy as a framework for measuring the relative power of models of hypercomputation comes from Post's theorem [44], which relates the levels of the arithmetical hierarchy to the degree structure arising from Turing-reducibility.

Theorem 3.3.4 (Post's theorem)**.** *Let $A \subseteq \mathbb{N}$. Then for all $n \in \mathbb{N}$ and all $B \subseteq \mathbb{N}$,*

$$B \in \Delta_{n+1}^A \Leftrightarrow B \leq_T A^{(n)}.$$

For a proof of Post's theorem, we refer to [26]. The following corollary follows immediately from Theorem 3.3.4.

Corollary 3.3.1. *Let $B \subseteq \mathbb{N}$. Then for all $n \in \mathbb{N}$,*

$$B \in \Delta_{n+1}^0 \Leftrightarrow B \leq_T \emptyset^{(n)}.$$

Measuring hypercomputational power

The arithmetical hierarchy can be used as a framework to measure and compare the relative power of hypercomputational models. The power is measured in terms of the sets that can be decided by each model. Ord [42] assesses the power of a selection of hypercomputational models, among which the o-machine, and asynchronous networks of Turing machines. We extend this assessment by including Turing machines with advice.

The hypercomputational power of models that use an external information resource, such as an oracle, advice, or timing function, depends on the degree of unsolvability of this source. An o-machine with a $\emptyset$ or Turing-computable oracle can not compute non-Turing-computable functions, since consulting such an oracle can be compared to making a subroutine call to another Turing machine. The same holds for asynchronous networks of Turing machines with synchronous or Turing-computable timing functions [42]. O-machines with an $\emptyset^{(n)}$ oracle can compute Δ_{n+1} functions, as can Turing machines with $\emptyset^{(n)}$ advice.

Van Leeuwen and Wiedermann's [68, 69] lineages of cognitive transducers and communities of active cognitive transducers are interactive models of computation, which compute translations over infinite symbols streams instead of functions from finite strings to finite strings. We explore the possibility to measure the power of interactive models by the streams they can produce relative to an interaction partner of a certain degree of unsolvability, comparable to the streams producible by o-machines relative to an oracle of a certain degree of unsolvability. In Section 3.2, we showed that if a stream $y \in \Gamma^\omega$ is interactively producible with respect to an interaction partner E, then y is producible by a comparable o-machine with comparable advice Θ^E. We argued that if a stream y is interactively producible with respect to an interaction partner E, it is Turing-reducible to Θ^E. Now we can apply Post's Theorem and state that if y is interactively producible with respect to environment E and

$\Theta^E \in \emptyset^{(n)}$, then $y \in \Delta^0_{n+1}$. This could serve as an upper bound to the power of interactive Turing machines.

Conjecture 3.3.1. *If $y \in \Gamma^\omega$ is interactively producible with respect to environment E and the oracle comparable to E — Θ^E — is in $\emptyset^{(n)}$, then $y \in \Delta^0_{n+1}$.*

Proof. Suppose y is interactively producible by interactive Turing machine M with respect to environment E. Then by Lemma 3.2.3, y is also producible by a comparable o-machine M' with comparable oracle Θ^E. Then $y \leq_T \Theta^E$. Since Θ^E is assumed to be in $\emptyset^{(n)}$, also $y \leq_T \emptyset^{(n)}$. By Post's Theorem, then $y \in \Delta^0_{n+1}$. □

Corollary 3.3.2. *The set of streams producible by interactive Turing machines with an $\emptyset^{(n)}$ environment $\subseteq \Delta^0_{n+1}$.*

Van Leeuwen and Wiedermann [68, 69] argue that lineages of cognitive transducers and communities of active cognitive transducers are as powerful as interactive Turing machines with advice. These interactive Turing machines with advice are non-uniform interactive models of computation. We propose to compare the streams producible by interactive Turing machines with advice to those producible by o-machines with two oracles: the first oracle providing the information provided by the interaction partner of the interactive Turing machine with advice, and the other oracle providing the information of the advice. Then, we conjecture that the power of o-machines with two oracles A_1 and A_2 depends on the degree of unsolvability of the most unsolvable oracle. If this conjecture is true, o-machines with two Turing-computable oracles can not produce non-Turing-computable output streams. In general, o-machines with $\emptyset^{(n)}$ and $\emptyset^{(m)}$ oracle can produce Δ_{k+1} output streams, where $k = \max(n, m)$. This adheres to saying that interactive Turing machines with advice with $\emptyset^{(n)}$ interaction partner and $\emptyset^{(m)}$ advice can produce Δ_{k+1} output streams, where $k = \max(n, m)$. Since lineages of cognitive transducers and communities of active cognitive transducers are equally powerful as interactive Turing machines with advice, lineages of cognitive transducers with $\emptyset^{(n)}$ interaction partner and $\emptyset^{(m)}$ evolution function, and communities of active cognitive transducers with $\emptyset^{(n)}$ interaction partner and $\emptyset^{(m)}$ description function can produce Δ_{k+1} output streams.

3.4 Realisability and exploitability

In Section 3.2, we discussed a selection of resources used by models of computation in order to gain hypercomputational power. These models are theoretical ones, and the resources they use, as well as the objects they manipulate, are abstract ones. We refer to such models capable of hypercomputation as 'hypercomputational models'. An interesting question is whether these hypercomputational models are physically realisable. In other words, whether nature allows for the existence or construction of physical systems that implement the hypercomputational models. We refer to physical systems capable of hypercomputation as 'hypercomputational systems'. Another question concerns the exploitability of the hypercomputational power of such systems; whether or not their hypercomputational power can be used to solve some predefined non-Turing-computable problem.

The physical realisability of hypercomputational models and the exploitability of their hypercomputational power is subject to debate. In fact, the issues seem to be at the heart of the controversy around hypercomputation. Ord [43, p. 144] states that "[t]he question whether or not we can physically compute more than the Turing machine is of fundamental importance and very much open". He [42] points out that claims about the physical realisability and exploitability are in fact claims about the nature of physics. Therefore, he adds, an assessment of the physical realisability of hypercomputational models and the exploitability of their hypercomputational power can only be based on current physics. According to Ord [43, p.144], "there are several places within current physical theories that hypercomputational processes might be found and there is little argument in the literature to show that of all the myriad ways that physical processes might combine, none of them will be hypercomputational". Davis [21, 22], on the other hand, argues that physical hypercomputation seems to be impossible according to currently accepted physical theories. For Davis [21, 22], the infeasibility of exploitable hypercomputation with respect to current physics is even reason to argue that "there is no such discipline as hypercomputation" [22, p. 4]. Stannett [53, p. 8] remarks that physical realisability is "in a sense a secondary issue", since "even if we accepted hypercomputation as having no basis whatsoever in physical reality, it is nonetheless an eminently useful *logical* idea, which offers a more comprehensive model of mathematical, physical and biologic processes than its merely computational counterpart". According to Copeland and Proudfoot, "'hypermachines' can be described on paper, but no one as yet knows whether it will be possible to build one" [18, p. 101]. It is clear that, up to the present, there is no generally accepted answer to the question of physical realisability and exploitability of hypercomputation.

In this section, we discuss issues concerning the physical realisability and exploitability of hypercomputational models with respect to the resources they use. In this discussion, we focus on resources used by Van Leeuwen and Wiedermann's lineages of cognitive transducers and communities of active cognitive transducers. For an assessment of a selection of other resources, such as infinite memory, and fair non-determinism, with respect to physical realisability and exploitability, we refer to [42].

External information resources

First, we discuss the issues of realisability and exploitability with regard to hypercomputational models making use of an external information resource. These models, among which o-machines, and Turing machines with advice are basically standard machines, like Turing machines. Hence, these models are physically realisable as far as Turing machines are. Technically, the external resources they use are not part of the models. However, whether or not the physical implementation of these models are also capable of hypercomputation depends on whether nature provides for the external resources they use. The existence or constructibility of physical implementations of non-Turing-computable information sources is crucial to the hypercomputational power of the systems corresponding to these models. Ord [42] points out that in order for physical systems implementing such models to be able to perform hypercomputation, the systems must have access to an external non-Turing-computable information source in nature.

According to Ord [42], candidates for such physical non-Turing-computable information sources are real valued quantities, as well as non-Turing computable processes. He states that since there are uncountably many real numbers, but only countably many Turing-computable real numbers, it is quite plausible that quantities exist in nature which take non-Turing computable values. Unfortunately, Ord adds, accessing such a non-Turing-computable real value requires arbitrary precision measurement, which seems impossible according to current physical theories. Davis [21] is pessimistic about the existence of non-Turing-computable values in nature. He states that "twentieth century physics has tended to see physical quantities as made up of discrete units" [21, p. 206].

The other approach, according to Ord [42], would be to observe some non-Turing-computable physical process, and use it in the computation. He points out that quantum mechanics suggests that random processes exist, and these processes are non-Turing-computable with probability one. However, Ord adds that even if processes existed in nature which are completely random in the sense of probability theory, these processes would be completely useless in the computation of a non-Turing computable mathematical function. Ord states that the existence of other non-Turing-computable processes, such as processes that could be used to produce for example τ — the real number corresponding to the halting set — is not physically impossible. On the other hand, he adds, we could not know for certain that we were actually dealing with the required process for our problem. Checking if we really dealt with, say τ, would require us to generate τ in the first place, and then check whether the process indeed corresponds to τ bit by bit. Davis [21] argues that even if one had access to a non-Turing-computable real number, "in order to use it as an oracle, one would also have to know its degree of unsolvability" [21, p. 207]. Ord explains that the best way would be if our best physical theory predicts this process to be the required one. He concludes that the ability to justify scientifically that using the resource gives the intended results is indeed a condition for all potential hypercomputational information resources to be harnessable. Davis states that "[i]n any case, a usable physical representation of an uncomputable function, would require a revolutionary new physical theory, and one that would be impossible to verify because of the inherent limitations of physical measurement" [21, p. 207]. Although the existence of non-Turing-computable quantities and processes in nature is not implausible, Cooper and Odifreddi [15] and Cooper [14] note that there is currently no evidence for the existence of non-Turing-computability in nature.

Infinite operation

Second, we consider the resource of infinite operation. Models using infinite operation as a resource, such as ω-machines and interaction machines, are basically classical models. Therefore, they are physically realisable as far as Turing machines are. However, their hypercomputational power relies on infinite computation time. For systems based on models which compute functions from finite inputs to finite outputs using infinitely many computational steps, one cannot wait infinitely long for the result of the computation. From systems based on models which translate infinite streams to infinite streams, in finite time, only a finite part of the translation can be finished. The only way for their hypercomputational power to be exploitable seems to be to observe the results of infinitely many computational steps in finite time. Hogarth [27] argues that there are relativistic space-times which allow an observer to view the eternity of the computer's computational process in finite time. Malament-Hogarth space-times, as such space-times are called now, are predicted to occur around black holes.

The accelerated Turing machine explicitly performs infinite computations in finite time by performing each time step in half the time used for the previous step. Ord [42] points out that this acceleration is highly problematic. One possibility would be for the tape head to move faster and faster with each step, which means that it eventually must go faster than the speed of light. As Ord [42] explains, this is at the edge of physical plausibility. Another possibility would be to decrease the distance that the head needs to move at each step [20]. However, this approach requires infinite spatial precision, which according to Ord [42] seems to be in conflict with quantum mechanics.

Infinite specification

In the infinite state Turing machine, the distinction between a classical part and an external resource cannot be made. Infinite state Turing machines have an explicit. As Ord [42] points out, specifying an infinite state Turing machine to compute an arbitrary function from $\mathbb{N}$ to $\mathbb{N}$ seems at least as hard as computing the function itself. On the other hand, if an infinite state Turing machine would be physically realisable, then its hypercomputational power would be exploitable.

Decide Δ_1^0 sets/ Produce Δ_1^0 streams	
Turing machines	
o-machines	Turing-computable oracle [42]
TM/A's	Turing-computable advice
asynchronous networks of TM's	synchronous or Turing-computable timing function [42]
interactive Turing machines	with interaction partner whose comparable oracle is Turing-computable (conjecture)
interactive Turing machines with advice	with Turing-computable interaction partner and Turing-computable advice (conjecture)
lineages of cognitive transducers	Turing-computable interaction partner and Turing-computable evolution function (conjecture)
communities of active cognitive transducers	Turing-computable interaction partner and Turing-computable description function (conjecture)

Decide Δ_{n+1}^0 sets/ Produce Δ_{n+1}^0 streams	
o-machines	$\emptyset^{(n)}$ oracle [42]
TM/A's	$\emptyset^{(n)}$ advice
interactive Turing machines	with interaction partner whose comparable oracle is $\emptyset^{(n)}$ (upper bound, conjecture)
interactive Turing machines with advice	with $\emptyset^k$ interaction partner and $\emptyset^l$ advice, where $\max(k, l) = n$ (upper bound, conjecture)
lineages of cognitive transducers	$\emptyset^{(k)}$ interaction partner $\emptyset^{(l)}$ evolution function where $\max(k, l) = n$
communities of active cognitive transducers	$\emptyset^{(k)}$ interaction partner $\emptyset^{(l)}$ description function where $\max(k, l) = n$

Table 3.1: The relative computational powers of models.

Chapter 4
Artificial life

In [69, 68], Van Leeuwen and Wiedermann study the computational power of artificial living systems. The study of artificial living systems is called 'Artificial Life' or 'ALife'. ALife is a relatively new field of research, which encompasses a broad range of research efforts. We do not go into detail on examples of ALife here, but rather discuss some of its central concepts and main methods. The aim of this discussion is to better understand the scope of Van Leeuwen and Wiedermann's results, as well as the implications of these results for ALife.

Bedau [5, p. 595] very generally refers to ALife as "an interdisciplinary study of life and life-like processes". Bonabeau and Theraulaz consider ALife as "a general method consisting in generating ... behaviors that are *interpretable as lifelike*" [6, p. 303]. They add that although this definition applies to almost everything that is done within the framework of ALife, "depending on the field to which it applies, this framework leads to very different results. For example, biologists do not have the same vision of artificial life as, say, computer scientists, artificial intelligence (AI) researchers, engineers or even artists" [6, p. 303]. Although there is an overlap with biology, Langton points out that ALife [34, p. 1] "complements the traditional biological sciences concerned with the *analysis* of living organisms by attempting to *synthesize* life-like behaviors within computers and other artificial media". In Section 4.1, we discuss the characteristics of living systems, and in Section 4.2, we discuss synthesis as the main method of ALife.

There are mainly two approaches to ALife [54]: the modelling and the engineering approach. The modelling approach aims at constructing systems that accurately model living systems. These models can then be used to test hypotheses regarding the behaviour of living systems. In this context, Bedau writes that ALife "attempts to understand living systems by artificially synthesizing extremely simple forms of them", whereby "it focuses on the essential rather than the contingent features of living systems" [5, p. 595]. The engineering approach aims at designing systems to accomplish some given complex task, inspired by the way natural systems accomplish it. Langton [36, p. x] explains that "[n]ature has discovered ingenious solutions to many hard engineering problems, problems that we have not been able to solve by our traditional engineering methods. The synthetic process of attempting to recreate these biological solutions in other materials will be of great practical use".

Based on the medium used for synthesising, ALife is usually divided into three approaches (see e.g. [5]): 'wet' ALife is concerned with the creation of life-like systems using biochemical materials, 'soft' ALife involves simulating life-like behaviour on a computer or other purely digital constructions, and 'hard' ALife creates hardware implementations of life-like systems. In soft ALife computers are used as a tool for synthesising life, and therefore there is an interesting link between ALife and the theory of computation. Langton [34] points out that

ALife requires a computational paradigm that differs from the standard Turing machine paradigm in a number of ways. We discuss this paradigm, as well as a model that is exemplary for this paradigm — the cellular automaton — in Section 4.3. In Section 4.4, we discuss hard ALife with a focus on concepts like embodiment and situatedness, and the role of a real physical environment.

4.1 Living systems

ALife is characterised by Langton as "the study of man-made systems that exhibit behaviors characteristic of natural living systems" [34, p. 1]. A natural living system is a multilevel phenomenon. Taylor and Jefferson [54, p. 1] explain that "[n]atural life on earth is organized into at least four fundamental levels of structure: the molecular level, the cellular level, the organism level, and the population-ecosystem level". Similarly, Bedau [4, p. 506] states that "[l]ife exhibits complex adaptive behavior at many different levels of analysis: metabolic and genomic networks, single cells, whole organisms, social groups, evolving ecologies, and so forth". Taylor and Jefferson add that "[a] living thing at any of these levels is a complex adaptive system exhibiting behavior that emerges from the interaction of a large number of elements from the levels below" [54, p. 1]. A complex system is "a network of interacting objects, agents, elements, or processes that exhibit a dynamic, aggregate behaviour" [6, p. 305]. Complex adaptive systems are special cases of complex systems, in which the elements adapt or learn as they interact [29]. That is, in complex adaptive systems, "the rules governing the elements are reshaped over time by some process of adaptation or learning" [4, p. 506]. Complex adaptive systems are central to artificial life, and ALife models reflect the complex adaptive nature of natural living systems.

4.2 Synthesis

Synthesis is the procedure of combining elements or components into a whole, as opposed to analysis, which is the procedure of breaking down a whole into parts or components. According to Bonabeau and Theraulaz, most of the work that has been done in ALife is based on the assumption that "synthesis is the most appropriate approach to the study of complex systems in general and of living complex systems in particular" [6, p. 303]. They add that synthesis "seems to be a good candidate, if not the only one, to explore the behavioral space of complex systems" [6, p. 305]. Bonabeau and Theraulaz [6] explain that it seems harder to start from manifestations of life, and try to find its basic underlying principles by top-down analysis than to start from simulations and try to synthesise behaviours of increasing complexity, which might eventually capture some aspects of life. Rather than top-down, models in ALife are therefore typically constructed from the bottom up by synthesis. This is also a fundamental difference between the approach of traditional symbolic AI and ALife in studying complex natural phenomena. As Bedau [5, p. 597] points out, "[m]ost traditional AI models are top-down-specified serial systems involving a complicated, centralized controller that makes decisions based on access to all aspects of global state. The controller's decisions have the potential to affect directly any aspect of the whole system". Artificial life models, on the other hand, reflect the distributed, parallel, and interactive nature of many natural living systems. As Bedau [5, p. 597] explains, alife models "are bottom-up-specified parallel systems of simple agents interacting locally. The models are repeatedly iterated and the resulting global behavior is observed ... The whole system's behavior is represented only indirectly. It arises out of interactions among directly represented parts ('agents' or 'individuals') and their physical and social environment".

The synthetic method of ALife allows for studying "*life-as-it-could-be*", rather than restricting the study to "*life-as-we-know-it*" [34, p. 1]. Langton [36, p. x] states that "[t]he set of biological entities provided to us by nature, broad and diverse as it is, is dominated by accident and historical contingency. We trust implicitly that there were lawful regularities at work in the determination of this set, but it is unlikely that we will discover many of these regularities by restricting ourselves only to the set of biological entities that nature actually provided us with. Rather, such regularities will be found only by exploring the much larger set of possible biological entities ... This is the role of synthesis, and this is the primary motivation for the field of Artificial Life: to give us a glimpse of that wider space of possible biologies".

However, Bonabeau and Theraulaz [6] point to the drawbacks of the synthetic approach in ALife. They explain that life-as-it-could be is ill-defined, and without taking into account empirical constraints observed by higher-level sciences, synthesis leads to exploration of *all* behaviours allowed by the exploration. This space of exploration then becomes immense. Moreover, Bonabeau and Theraulaz explain that the models constructed by synthesis have weakened explanatory status for the phenomenon or behaviour they reproduce. They refer to the distinction made by Putnam [46] between *deducing* and *explaining* properties of a phenomenon from a set of causes. To explain the phenomenon is to determine the relevant causes. Although for certain systems the microstructure might be largely irrelevant, the number of remaining relevant causes might also be very high.

4.3 Soft ALife

Synthesis is the main method in the field of ALife, but different media can be used for synthesis. The media used are often divided in three categories: biochemical materials, digital computers, and hardware. In 'soft ALife' ('soft' from 'software'), the computer is used as a tool for synthesising life-like systems. The use of computers points to a relation between soft ALife and the theory of computation. In this section, we discuss this relation in more detail. Then, we discuss a prototypical example of soft ALife: the cellular automaton. The cellular automaton is a well-studied model, and its behaviour has been characterised into four classes by Wolfram [72]. These classes of cellular automaton behaviour have been related to results from computability theory.

Soft ALife and computation

Langton [34] points out that the work of Gödel, Church, Kleene, Turing, and Post in the theory of computation led to the realisation that "the essence of a mechanical process — the 'thing' responsible for its dynamic behavior — is not a thing at all, but an abstract control structure, or 'program' — a selection of simple actions from a finite repertoire", the essential features of which "could be captured within an abstract set of rules — a formal specification — without regard to the material out of which the machine was constructed" [34, p. 11]. The universal Turing machine provides an insightful illustration of this; it receives a formal description of a Turing machine M as input, and simulates M on provided input. In this way, the behaviour of M is produced without actually building a machine for M. General purpose computers are machines that resemble the universal Turing machine in this respect.

Machines can thus be thought of in terms of their formal description. Reversely, Langton explains, "we can ... view abstract, formal specifications as potential machines" [34, p. 11]. He argues that "[i]n mapping the machines of our common experience to formal specifications, we have by no means exhausted the space of *possible* specifications. Indeed, most of our individual machines map to a very small subset of the space of specifications — a subset largely characterized by methodical, boring, uninteresting dynamics. When placed together in aggregates, however, even the simplest machines can participate in *extremely* complicated dynamics" [34, p. 11]. Here, general purpose computers come in as a useful tool, since they can be given a program, or formal description, and simulate the corresponding machine.

However, work in the theory of computation has also revealed the intrinsic limitations of universal Turing machines, and hence of general purpose computers that operate under the scenario of the Turing machine. Langton [34] points out that the behaviours that can be generated by general purpose computers are limited in two ways. First, for some behaviours, it is not possible in principle to give an algorithm that produces that behaviour. These are the behaviours that are non-Turing-computable. An example of a non-Turing-computable function is the halting function we mentioned in Chapter 3. The non-Turing-computability of the halting function is a special case of a more general result by Rice [47]. Informally, Rice's theorem states that it is not in general effectively decidable whether the function ϕ_M computed by some Turing machine M has a certain non-trivial property. A non-trivial property is a property that is held by at least one, but not all Turing-computable functions. Rice's theorem states that non-trivial properties of the future behaviour of an algorithm cannot be predicted in an effective way. Second, Langton [34] points out that there are behaviours for which we do not know an algorithm to produce it, even though such an algorithm may exist in principle. Therefore, he explains, "[w]e need to separate the notion of a formal specification of a machine ... from the notion of a formal specification of a machine's behavior — that is, a specification of the *sequence of transitions* that the machine will undergo. We have formal systems for the former, but not for the latter. In general we can neither derive behaviors from specifications nor derive specifications from behaviors" [34, p. 12]. This is an important insight for ALife, since it entails that for some machines, in order to determine their behaviour, the only option is to run the machine and observe their behaviour. Langton adds that "[t]his has consequences for the methods by which we (or nature) go about *generating* behavior generators themselves" [34, p. 12].

Although general purpose computers are used as a tool for synthesising life-like behaviour in ALife , the computational paradigm of the Turing machine is not suitable for ALife. Langton [34, p. 3] points out that ALife requires a new approach to computation: "one that focuses on *ongoing dynamic behaviour* rather than on any *final result*". It is this ongoing dynamics, or behaviour of the system at hand that is interesting to ALife, not the ultimate state reached by that dynamics. Langton [34] explains that most historical attempts to build imitations of living things involved a central program which was responsible for the dynamical behaviour of the model. However, he argues, "[t]he most promising approaches to modeling complex systems like life or intelligence are those which have dispensed with the notion of a centralized global controller, and have focused instead on mechanisms for the *distributed* control of behavior" [34, p. 21]. Langton identifies some essential features to computer-based alife models. First, they should consist of populations of simple programs. Second, there should be no program controlling all of the other programs. Third, each program should specify the way in which a simple entity reacts to local situations in its environment, including interactions with other entities. Finally, there should be no rules that dictate global behaviour. As a result, any behaviour at levels higher than the individual programs should be emergent.

Cellular automata

A prototypical model of the computational paradigm suitable for soft ALife is the cellular automaton (CA). This model of computation is originally due to Van Neumann [65] by Ulam's suggestion. According to Wolfram [72, p. 1] "Cellular automata are mathematical models for complex natural systems containing large numbers of simple identical components with local interactions". Langton [34, p. 13] states that "CA's are good examples of the kind if computational paradigm sought after by Artificial Life: bottom-up, parallel, local-determination of behavior".

A standard cellular automaton consists of a regular lattice of cells $\mathcal{L}$, each of which contains an identical deterministic finite state automaton. At any time, each finite state automaton can be in only one of a finite number of states. A neighbourhood relation is defined over the lattice, assigning to each cell a set of neighbour cells. At each time, every finite state automaton makes a state transition according to its transition function. Thereby, it takes as input the states of all cells in its immediate neighbourhood, including its own state. The state of the CA at time t is the collection of states of each of its components.

At time $t = 0$, the initial state of the CA must be given. In one time-step, all finite state automata residing at the cells are updated. Since each of the finite state automata behaves deterministically, the evolution of the CA is also deterministic. In fact, since each of the finite state machines has only a finite set of possible states, say Q, a CA with n cells can enter only in $|Q|^n$ different states. The set of possible states a CA can enter is called its 'state space'. The evolution of a cellular automaton can be studied by observing the sequence of configurations it goes through in its state space. Note that it is the *state* of the CA that changes over time, not the structure of the CA.

Wolfram [72] studied cellular automata in a systematic way and found empirical evidence for the existence of four qualitative classes of cellular automaton limiting behaviour (see Table 4.1). In [71], Wolfram relates this classification to characterisations in the stability or predictability of the cellular automata behaviour under small perturbations in their initial configurations. Wolfram's classes and their stability and predictability characteristics are shown in Table 4.1. Langton [35] makes a connection between Wolfram's classes of automata and the halting problem. He explains that there are three possibilities for the ability to decide the outcome of a computation: we are able to determine that the computation eventually halts, or that it will never halt, or we are not able to determine whether or not it will halt. These three possibilities, Langton continues, are reflected in the possibilities for the ultimate outcome of cellular automaton behaviour. Class I and II automata "'freeze up' into short-period behavior from any possible configuration" [35, p. 33]. Class III automata, on the other hand, "will never freeze into periodic behaviour" [35, p. 33]. For Class IV automata it "will be 'effectively' undecidable whether a particular rule operating on a particular initial

Class	Limiting behaviour	Stability	Predictability
I	Evolution to a homogeneous state	No change in final state	Entirely predictable, independent of initial state
II	Evolution to a set of simple stable or periodic states	A change of a cell state in the initial configuration only affects the final cell state of cells relatively nearby the changed cell	Local behaviour is predictable from local initial state
III	Evolution to a chaotic, aperiodic configuration	A small change in the initial state leads to changes over an ever-increasing region	Behaviour depends on ever-increasing initial region
IV	Evolution to complex localised patterns, which are sometimes propagating	Irregular changes	Effectively unpredictable

Table 4.1: Wolfram classes of cellular automata, and their characteristic limiting behaviour, stability under small perturbations, and predictability.

configuration will ultimately lead to a frozen state or not" [35, p. 33].

In [11], Cook proves Wolfram's [72] conjecture that cellular automata are capable of universal computation. In fact, Cook proves this for one of the most simple one-dimensional cellular automata. The implication of this result is that many questions concerning the behaviour of cellular automata, such as whether the CA will reach a stable state in the limit for arbitrary initial condition, are in general undecidable.

4.4 Hard ALife

In the previous section, we discussed soft ALife, which is concerned with synthesising life-like behaviour using digital computers as a medium. Hard ALife , on the other hand, its concerned with physical artificial living systems. Much of the work in hard ALife involves autonomous agents, or robots. According to Bedau, "[h]ard artificial life tries to synthesize autonomous adaptive and intelligent behavior in the real world ... by exploiting biological inspiration whenever possible" [5, p. 600]. Hard ALife is especially suitable for the relatively new approach of 'adaptive autonomous agents', although Maes [37] notes that this approach is not necessarily restricted to hardware forms of ALife.

Adaptive autonomous agents

Research in adaptive autonomous agents is also called 'behaviour-based AI', or 'bottom-up AI'. Adaptive autonomous agents are "systems that inhabit a dynamic, unpredictable environment in which they try to satisfy a set of time-dependent goals or motivations" [37, p. 135]. Maes [37] identifies some key points that distinguish traditional AI, or 'top-down AI', from the research in autonomous agents. One of these key points is that mainstream AI focuses on 'closed' systems. These systems do not directly interact with their problem domain — their environment — but only in an indirect and controlled way via a human operator. Autonomous agents, on the other hand, are 'open systems', which are 'situated' in their environment. This means that an agent has sensors and actuators through which it is directly connected to its problem domain. Actuators enable the agent to affect its problem domain. This domain is typically dynamic, unpredictable, and possibly incorporates other natural or artificial agents. The dynamic nature of the environment puts time constraints on actions or reactions of autonomous agents.

Another key difference identified by Maes [37] is that traditional AI systems typically deal with one problem at a time, and that the system shuts out the environment while solving the problem. This means that "from the system's point of view, the problem domain does not change while the system is computing" [37, p. 137]. In contrast, autonomous adaptive agents have to monitor their environment for new problems and goals, and possibly deal with conflicting goals simultaneously.

Moreover, Maes [37] points out that traditional AI is typically not concerned with how systems have to adapt over time to changing situations. In autonomous agents research, on the other hand, adaptation is a central focus point. Either the agents improve their own structure, and hence their behaviour over time based on experience, or the designer evolves gradually adds structure to the existing system to evolve it into a more sophisticated system.

Maes [37] discusses two important insights in which research in autonomous agents is grounded. The first insight is that "[l]ooking at complete systems changes the problems often in a favorable way" [37, p. 139]. Among other things, complete systems are situated in an environment. According to Bedau [5], the fact that hardware agents are embodied as well as situated in a real environment yields an advantage for hard ALife. The physical environment can be exploited for generating autonomous, adaptive, and intelligent behaviour. Bedau explains that "[o]ne of the tricks is to let the physical environment be largely responsible for generating the behavior. Rather than relying on an elaborate and detailed internal representation of the external environment, the behavior of biologically-inspired robotics quite directly depends on the system's sensory input from its immediate environment. With the right sensory-motor connections, a system can quickly and intelligently navigate in complex and unpredictable environments" [5, p. 600]. Moreover, Maes [37] points out that agents are typically part of a society, composed of other agents, solving similar problems in the same environment. This allows agents to make use of the problem solving abilities of other agents, for example by observing or imitating others.

The second insight is that "[i]nteraction dynamics can lead to emergent complexity [37, p. 139]. This means that "the internal structures controlling an agent need not be complex to produce complex resulting behavior" [37, p. 140]. In bottom-up generated complexity, none of the components is primarily responsible for the complex behaviour. Therefore, this emergent complexity is often more robust than top-down organised complexity. Moreover, Maes [37] argues that systems consisting of parallel, interacting components are able to adapt more quickly to the dynamical environment than sequential systems, since multiple solutions can be explored in parallel.

Chapter 5
Emergence

The third and last notion present in Van Leeuwen and Wiedermann's [68, 69] claims is the notion of emergence. In order to review Van Leeuwen and Wiedermann's claims more critically, we discuss emergence in more detail in this chapter. Although the idea of emergence probably can be traced back to ancient philosophy, the concept was first worked out comprehensively by a number of British scientists in the late nineteenth and early twentieth century, among which Mill [38], Broad [7], Morgan [40], and Alexander [1]. These scientists and their view of emergence are nowadays referred to as 'British emergentists' and 'British emergentism' respectively. For a brief history of British emergentism, we refer to [41]. Since then, the term 'emergence' has been used in various contexts, for different purposes, and with different meanings. This has become even more apparent now that there is renewed interest in emergence within in the field of complex system theory. Kim [33, pp. 547-548] nicely verbalises the lack of consistency in the use of the term emergence that has come with its recent growth in popularity when he states that "[w]e now see the term being freely bandied about ... with little visible regard for whether its use is underpinned by a consistent, tolerably unified, and shared meaning". Cooper [13, p. 194] writes that "[f]or the most part, the emergence of *Emergence* as something about which everyone has something to say, has generated more questions than answers, and more excitement than clarity". Ronald, Sipper and Capcarrère even observe that "overly facile use of the term emergence has made it controversial" [48, p. 225]. According to Cooper [14, p. 1352], "[e]mergence lies at the core of a number of controversies in science".

As appears from the above statement, there is no generally accepted notion of emergence. However, before discussing some of the existing notions of emergence, we try to pin down a number of its general characteristics. According to Schröder [50], things, properties, and laws can be labelled 'emergent'. However, he adds, the most basic class of the three seems to be the class of emergent properties, as a thing can be emergent only if it has at least one emergent property, and a law can be considered emergent if it connects emergent properties. According to Schröder, emergent properties are generally considered the properties of complex systems, not properties of elementary particles. Moreover, in general, in order to be considered emergent, a property of a complex thing must not be a property of a proper part of that thing. O'Connor and Wong [41] give a general definition of emergent entities as entities that "'arise' out of more fundamental entities and yet are 'novel' or 'irreducible' with respect to them". According to Kim, "the intuitive idea of an emergent property stems from the thought that a purely physical system, composed exclusively of bits of matter, when it reaches a certain degree of complexity in its structural organization, can begin to exhibit genuinely novel properties not possessed by its simpler constituents" [33, p. 548]. However, Kim adds, in this idea it is not yet made precise what is meant by a 'novel' property. It is exactly in making the idea of what emergence is more precise that disputes arise.

In Section 5.1, we first introduce Humphreys' [30] twofold taxonomy of emergence. Then, we discuss two notions of emergence in more detail: a version of strong emergence by Kim [33, 32] in Section 5.2, and Bedau's [3] notion of weak emergence in Section 5.3. Strong emergence [33] has significant continuity with the traditional, British emergentist view of emergence. Weak emergence [3] is specifically applicable in the context of complex systems. Therefore, it is especially relevant to ALife, and our discussion of Van Leeuwen and Wiedermann's claim.

5.1 A relational and temporal taxonomy

Humphreys [30] presents a relational, as well as a temporal taxonomy of emergence. The relational taxonomy is based on the relation between the emergent entity and the entities from which it emerges. The taxonomy has three divisions: the inferential approach, the conceptual approach, and the ontological approach. In the inferential approach, "an entity ... is emergent with respect to a domain D if and only if it is impossible, on the basis of a complete theory of D, to effectively predict that entity or to effectively compute a state corresponding to that feature" [30, p. 584]. According to Humphreys, one of the examples of computationally based treatments of the inferential approach is weak emergence, which we discuss below. In the conceptual approach, "an entity ... is conceptually emergent with respect to theoretical framework F if and only if a conceptual or descriptive apparatus that is not in F must be developed in order to effectively represent that entity" [30, p. 585]. In the ontological approach, "an entity is ontologically emergent with respect to domain D if and only if that entity is ontologically irreducible to entities in domain D" [30, p. 585]. Humphrey adds that the three approaches are not mutually exclusive. He stresses that in order for elements to fall into either one of these three categories, the elements ought to be relational. That is, the criteria for emergence are not applicable to an entity in isolation, but with respect to a domain D or framework F. The relevant relations between the emergent entity and the domain or framework determine the emergent status of the entity.

The temporal taxonomy has two divisions: diachronic emergence, and synchronic emergence. An emergent phenomenon is a case of diachronic emergence if it emerges from preceding phenomena due to a temporally extended process. Synchronic emergence, on the other hand, involves properties at a higher-level, which emerge from lower level properties existing at the same time.

5.2 Strong emergence

Kim [33] argues for significant continuity with the concept that British emergentists seem to have had in mind. He identifies two components that are necessary for any concept of emergence that is true to its historical origins: supervenience and irreducibility of emergents. Supervenience is defined in [33, p. 550] as follows.

Definition 5.2.1 (Supervenience/determination). *Property M supervenes on, or is determined by, properties $N_1, ..., N_n$ in the sense that whenever anything has $N_1, ..., N_2$ [sic], it necessarily has M.*

According to Kim [33], supervenience is the first condition for emergence. Kim explains that the second condition — irreducibility — requires more discussion. First, he defines two other relations a property can bear to a set of other properties: predictability and explainability. The following definitions are from [33, p. 550].

Definition 5.2.2 (Predictability). *The occurrence of M – that is, whether M will be instantiated on a given occasion – can be predicted from the occurrence of $N_1, ..., N_2$ [sic]; full information concerning whether $N_1, ..., N_n$ are instantiated in a system suffices for the prediction of whether the system instantiates M.*

Definition 5.2.3 (Explainability). *Why a system instantiates M can be explained, understood, and made intelligible in terms of its instantiating $N_1, ..., N_n$.*

According to Kim [33], British emergentists seemed to deny predictability or explainability of emergent properties from its basal conditions. However, in the above definitions, the notions of prediction and explanation remain vague and intuitive ones. This can be problematic. Kim [33] points out that if the 'emergence law' linking the phenomenon with facts at a basal level is included in the evidence base, in some sense, the occurrence of an emergent phenomenon can indeed be predicted inductively. The same holds for explainability. Nagelian 'bridge-laws', connecting properties to be reduced with base-level properties, are not allowed in reductive prediction or explanation. The very question of emergentists is why these particular bridge laws hold.

Kim [32] introduces a model to accommodate what he calls 'functional reduction'. Functional reduction proceeds via the process of 'functionalisation' of the property to be reduced in terms of properties at the base level. This process proceeds as follows. Let B be the reduction base, the domain of facts, properties, containing the basal conditions for emergent properties. Property E must be functionalised, that is, "construed ... as a property defined by its causal/nomic relations to other properties, specifically properties in the reduction base B" [32, p. 10]. Having E is then defined as having some property P in B such that $C_1, ..., C_n$ cause P to be instantiated, and P causes $F_1, ..., F_m$ to be instantiated. Any property P that satisfies this specification is a so called 'realiser' of E. Then, realisers of E are to be found in B through empirical scientific research. Finally, a theory is to be found at the level of B that explains how realisers of E perform the causal task that is constitutive of E. Kim [33] identifies irreducibility of emergents as the second condition of emergence.

(Condition 1. Supervenience). *If property M emerges from properties $N_1, ..., N_n$, then M supervenes on $N_1, ..., N_n$.*

(Condition 2. Irreducibility of emergents). *Property M is emergent from a set of properties, $N_1, ..., N_n$ only if M is not functionally reducible with the set of the Ns as its realiser.*

Although Kim argues that supervenience and irreducibility are indeed necessary components of emergence in the classical, British emergentism conception, he proceeds by arguing that these two components are not sufficient conditions of emergence. Actually, he states, the two conditions are essentially negative: "[t]hey tell us what emergence is not; they do not tell us anything – at least, not much – about what it is" [33, p. 557]. Kim adds that any positive characterisation of emergence should "explain why emergents so characterized supervene on their base properties and why, in spite of the supervenience relation, the former are not reducible to the latter; second, it must successfully cope with the problem of downward causation" [33, p. 557].

Kim [33] explains downward causation as follows. Suppose an instance of an emergent property M causes another emergent property M^* to instantiate. This is an instance of so called 'same-level causation'. As an emergent, M^* must have a basal property P^* from which it emerges. Since supervenience is a condition for emergence, M must supervene on P^*, which means that the presence of P^* itself guarantees that M^* will be instantiated at that time. That is, whenever P^* is present at that time, M^* will also be present at that time, regardless of whether M has been there at all. However, we assumed M to be the cause of M^*. The only way to save the claim that M caused M^* is to say that M caused M^* by causing P^*. However, since P^* is a property at the base level, this would entail 'downward causation' from M to P^*.

$$\begin{tikzcd}
M \arrow[r, "\text{causes}"] & M^*
\end{tikzcd}$$

(a) Same-level causation.

$$\begin{tikzcd}
M \arrow[r, "\text{causes}"] & M^* \\
& P^* \arrow[u, "\text{superv.}"']
\end{tikzcd}$$

(b) Supervenience.

$$\begin{tikzcd}
M \arrow[r, dashed, "\text{causes}"] \arrow[dr, "\text{causes}"'] & M^* \\
& P^* \arrow[u, "\text{superv.}"']
\end{tikzcd}$$

(c) Downward causation.

Figure 5.1: Downward causation.

$$\begin{tikzcd}
M \arrow[r, dashed, "\text{causes}"] \arrow[dr, "\text{causes}"] & M^* \\
P \arrow[u, "\text{superv.}"] & P^* \arrow[u, "\text{superv.}"']
\end{tikzcd}$$

(a) Supervenience.

$$\begin{tikzcd}
M \arrow[r, dashed, "\text{causes}"] \arrow[dr, "\text{causes}"] & M^* \\
P \arrow[u, "\text{superv.}"] \arrow[r, dashed, "\text{causes}"] & P^* \arrow[u, "\text{superv.}"']
\end{tikzcd}$$

(b) Causal overdetermination.

Figure 5.2: Problem with downward causation.

Kim points out the problem with downward causation. Since M is itself an emergent property, it must also have a base property, say P (see Figure 5.2(a)). If causation is understood as nomological sufficiency, P is nomologically sufficient for M, and M is nomologically sufficient for P^*. It follows that P is nomologically sufficient for P^*, and hence P qualifies as a cause for P^* (see Figure 5.2(b)). If both M and P retain as a cause of P^*, downward causation has lead to causal overdetermination. Moreover, it goes against the emergentist idea that emergents are to make distinctive and novel causal contributions.

5.3 Weak emergence

The notion of weak emergence, defined by Bedau [3], is a more "innocent form of emergence" [3, p. 375]. Bedau points out that whereas strong emergence suffers from the problem of downward causation by requiring emergents to have irreducible causal influences, weak emergence "involves downward causation only in the weak form created by the activity of the micro-properties that constitute structural macro-properties" [3, p. 376]. Moreover, Bedau argues that weak emergence is ubiquitous in complex systems, and is therefore especially relevant to science. Since natural and artificial living systems are regarded as complex systems, weak emergence is specifically relevant to ALife.

Weak emergence applies when there is a system S, which is composed out of a time-varying set of low-level parts. S has various 'microstates', which are the intrinsic states of its low-level parts. S also has various 'macrostates', which are structural properties fully constituted out of its microstates. There is a microdynamic, denoted by D, which governs the evolution of S's microstates over time. This microdynamic might be 'local' in the sense that it takes only the microstates of parts in a local neighbourhood into account when determining the new state of a low-level part. External conditions are those conditions that are "outside the system" [3, p. 378], affecting the system's microstates. Bedau [3, p. 378] defines weak emergence as follows.

Definition 5.3.1 (Weak emergence)**.** *Macrostate P of S with microdynamic D is* weakly emergent *iff P can be derived from D and S's external conditions but only by simulation.*

If D is deterministic and S is a closed system, the only external condition is the system's initial condition. However, Bedau adds, "[i]f the system is open, then another kind of 'external' condition is the contingencies of the flux of parts and states through S. If the microdynamic is non-deterministic, then each accidental effect is an 'external' condition" [3, p.378].

Given the initial condition of the system, as well as the stream of all other external conditions, one can simulate the system by iterating its microdynamic. Given the sequence of external conditions, the microdynamic completely determines the sequence of microstates the system goes through. Bedau points out that "[s]ince the macrostate P is a structural property constituted out of the system's microstates, the external conditions and the microdynamic completely determine whether P materializes at any stage in the simulation. By simulating the system in this way one can derive from the microdynamic plus the external conditions whether P obtains at any given time after the initial condition. What distinguishes a weakly emergent macrostate is that this sort of simulation is required to derive the macrostate's behavior from the system's microdynamic" [3, p. 287].

Bedau [3] points to a number of problems concerning weak emergence. First, a weakly emergent property can in principle be 'predicted' from information about the system's microdynamic and external conditions. This seems to go against traditional ideas of emergence. However, Bedau [3] points out that in case of open systems, such a 'prediction' is only possible in principle. In practice, he explains, we do not have prior knowledge of accidental changes resulting from interaction with the external world in the case of open systems. Moreover, he adds, even for closed, deterministic systems, 'prediction' by step-by-step iteration of the system to observe its behaviour over time can hardly be called prediction.

Second, Bedau explains that in general, we may not have proof that a given macrostate of a given system is not derivable without simulation. Hence, we cannot in general prove that a property is weakly emergent. However, Bedau argues that weak emergence claims can have substantial empirical support.

Chapter 6
Discussion

In [68, 69], Van Leeuwen and Wiedermann claim that hypercomputational power can emerge in artificial living systems. In this chapter, we review Van Leeuwen and Wiedermann's claim in more detail. Based on our previous account on hypercomputation, artificial life, and emergence, we discuss in Section 6.1 whether Van Leeuwen and Wiedermann's claim is indeed tenable. In Section 6.2, we point to some implications of Van Leeuwen and Wiedermann's results for the field of ALife.

6.1 Tenability

In this section, we review the tenability of Van Leeuwen and Wiedermann's [68, 69] claim that hypercomputational power can emerge in artificial living systems. This discussion is based on our reproduction of Van Leeuwen and Wiedermann's argument in Chapter 2, as well as our account on hypercomputation, artificial living systems, and emergence. First, we discuss whether the computational models proposed by Van Leeuwen and Wiedermann can indeed be considered artificial living systems. Then, we discuss whether Van Leeuwen and Wiedermann's models are indeed hypercomputational, and if so, whether we can give a more specific characterisation of their power. Moreover, we point out what the hypercomputational power of the models means for the computational power of artificial or natural living systems. Third, we examine whether Van Leeuwen and Wiedermann's results, restated here in Theorem 2.3.1 and Theorem 2.3.2, indeed involve a case of emergence, and if so, according to which notion.

Artificial life

First, we discuss whether Van Leeuwen and Wiedermann's [68, 69] results indeed say something about the computational power artificial living systems. In Chapter 2, we have seen that to measure the computational power of natural or artificial living systems, Van Leeuwen and Wiedermann present three computational models: the active cognitive transducer as a model of an individual organism, the lineage of cognitive transducers as a model of subsequent generations of organisms, or alternatively of the subsequent stages in the process of an individual organism's adaptation, and the community of active cognitive transducers as a model of a time-varying population of individuals. For Van Leeuwen and Wiedermann's claim to hold, it is important that their models are indeed accurate models of natural or artificial living systems. In this discussion, we focus on communities of active cognitive transducers.

In Chapter 4, we have seen that artificial living systems are man-made systems that exhibit life-like behaviour. One of the goals of the modelling approach to ALife is to find the essential features of living systems. Research in ALife is mostly based on the assumption that it is hard to find the underlying principles of life by analysing manifestations of life. Rather, the central method used in ALife to find the fundamental features of living systems is the synthesis of simple systems to generate of behaviours of increasing complexity. The formulation of computational models of natural or artificial living systems seems to go against this central assumption in ALife.

On the other hand, it is often held that to produce life-like behaviour, a system should at least share the main characteristics of complex adaptive systems; it should be composed of multiple interacting elements, which adapt as they interact, and which exhibit a dynamic, aggregate behaviour. Van Leeuwen and Wiedermann's communities of active cognitive transducers incorporate these characteristics of complex adaptive systems. At each time they are composed of a number of individual active cognitive transducers, which communicate, or interact with each other and the environment. At each time, the configuration of a community is described in terms of the configurations of its constituting parts at that time. Likewise, the input and output to and from the community is described in terms of the input and output to and from its constituting parts. Although the individual active cognitive transducers in a community are not assumed to adapt or evolve over time, the community as a whole evolves, since its composition changes over time. This scenario seems to include evolutionary scenario's that can be interpreted as evolving components. Therefore, it seems that Van Leeuwen and Wiedermann's communities of active cognitive transducers can be interpreted as models of complex adaptive systems.

In Chapter 4, we have seen that living systems are multi-levelled phenomena, and a living thing at each level is a complex adaptive system, consisting of multiple interacting elements, which change over time. This means that communities of active cognitive transducers need not necessarily be interpreted as models of systems on the population level, but can be interpreted as complex adaptive system on different levels of structure.

Although the class of communities of active cognitive transducers seems to correspond to the class of complex adaptive systems, this doesn't mean that all instances of communities of active cognitive transducers can automatically be considered artificial living systems as well. Whether or not a given instance of the class of communities of active cognitive transducers is an artificial living system can only be determined by finding out whether its behaviour can be considered life-like. Rice's theorem [47] states that even for Turing-equivalent systems, any non-trivial property of a system's behaviour is in general not effectively decidable by looking at its description alone; it can only be concluded after running the system and observing its behaviour. The same holds for properties of the behaviour of communities of active cognitive transducers, which can in general not even be simulated by standard Turing machines. Therefore, given an instance of communities of active cognitive transducers, determining whether or not it is an artificial living system requires running the system and observing its behaviour. The behaviour of a community of active cognitive transducers depends largely on its description function, which determines how the community evolves over time, and on its timing function, which has an influence on how the components communicate. The class of communities of active cognitive transducers may include a large number of communities of active cognitive transducers that exhibit non-interesting behaviour, which is either too simple, or too chaotic to be considered life-like.

Therefore, our first conclusion is that Van Leeuwen and Wiedermann's communities of active cognitive transducers seem to be accurate models of complex adaptive systems, which include natural and artificial living systems. Determining whether an instance of the class of communities of active cognitive transducers is an artificial living systems in general requires running the system and observing its behaviour. Consequently, Van Leeuwen and Wiedermann's results seem to be assertions about the computational power of complex adaptive systems in general, rather than of artificial living systems in specific.

Hypercomputation

Our second point of discussion concerns the hypercomputational power of Van Leeuwen and Wiedermann's [68, 69] models. More specifically, we first examine whether Van Leeuwen and Wiedermann's lineages of cognitive transducers and communities of active cognitive transducers are indeed hypercomputational models. Then, we discuss whether it is possible to give a more specific characterisation of the computational power of the models. Finally, we discuss what the hypercomputational power of lineages of cognitive transducers and communities of active cognitive transducers means for the computational power of their physical counterparts: complex adaptive systems.

Before we discuss whether Van Leeuwen and Wiedermann's models indeed possess hypercomputational power, we remark that the authors use the term 'super-Turing computing power' rather than 'hypercomputational power' to characterise the power of their models. The authors refer to a system possessing super-Turing computing power as a system that "can perform computational tasks that cannot be achieved by classical means, making use of the computational mechanism of standard Turing machines or its equivalents" [69, p. 205]. In this description of super-Turing power, Van Leeuwen and Wiedermann do not seem to require that the super-Turing system is also capable of computing the Turing-computable functions. Therefore, the set of computing systems applying to Van Leeuwen and Wiedermann's description of super-Turing systems may include systems that are 'non-Turing' according to Stannett's [52] definition. Using the terminology adopted in this thesis, the more inclusive term 'hypercomputational power' thus seems the most suited to the authors' informal description.

Van Leeuwen and Wiedermann's [68, 69] argument for the hypercomputational power of lineages of cognitive transducers and communities of active cognitive transducers proceeds by stating that the models are equally powerful as interactive Turing machines with advice. ITM/A's, in turn, are proven to be strictly more powerful than interactive Turing machines without advice. We reproduced Van Leeuwen and Wiedermann's statements in Theorem 2.3.1 and Theorem 2.3.2. Van Leeuwen and Wiedermann do not prove Theorem 2.3.2 in [68, 69]. Therefore, we developed our own proof for one way of this theorem in Section 2.3. From the results in Section 2.3, we conclude that lineages of cognitive transducers and communities of active cognitive transducers are indeed more powerful than ITM's without advice.

Still, for Van Leeuwen and Wiedermann's models to be super-Turing in Stannett's [52] sense, they must also be able to compute the Turing-computable functions. Van Leeuwen and Wiedermann [68, 69] do not make explicit how the power of interactive Turing machines with advice relates to that of standard, non-interactive Turing machines. In Section 3.2 we have seen that there seems to be no consensus on the power of interactive models of computation. Wegner [66] claims that interactive computation is in itself more powerful than non-interactive computation, such as the computation performed by standard Turing machines. Van Leeuwen and Wiedermann [63] suggest that since interactive Turing machines compute translations on the set of infinite streams rather than on the set of finite strings, their power is incomparable to that of standard Turing machines. Since interactive Turing machines with advice, lineages of cognitive transducers and communities of active cognitive transducers are also interactive models, this suggestion would imply that lineages of cognitive transducers and communities of active cognitive transducers are non-Turing rather than super-Turing in Stannett's [52] sense.

On the other hand, in Section 3.2 we proposed to view interactive Turing machines as models of relative computability, as suggested by Prasse and Rittgen [45]. Instead of the translations realisable by interactive Turing machines, we looked at the streams producible by interactive Turing machines relative to an interaction partner. This allows us to compare the streams producible by interactive Turing machines directly to those producible by standard Turing machines, as a special case of those producible by o-machines relative to an oracle. We conjectured that the set of infinite streams producible by interactive Turing machines relative to a given interaction partner is a subset of the set of streams producible by comparable o-machines with a comparable oracle. If this conjecture is indeed true, interactive Turing machines can themselves be seen as hypercomputational models. The set of streams producible by interactive Turing machines with advice relative to an interaction partner and an advice is then a subset of the streams producible by o-machines with two oracles: the first comparable to the interaction partner and the second to the advice.

We conclude that as conceptual models of computation, lineages of cognitive transducers and communities of active cognitive transducers indeed possess hypercomputational power, but that there seems to be no consensus on whether they are super-Turing or non-Turing in Stannett's [52] sense. Further research is needed to gain more insight in the power of interactive models of computation. One direction for further research could be to look at our proposal in Section 3.2 in more detail.

Having seen that Van Leeuwen and Wiedermann's [68, 69] models are indeed hypercomputational, we now discuss the possibility of giving a more specific characterisation of the computational power of the models. We consider both the framework of the arithmetical hierarchy, which we discussed in Section 3.3, and the cellular automaton based Wolfram classes [71, 72], which we discussed in Section 4.3.

In Section 3.3, we have seen that the world beyond the Turing-computable problems is hierarchically structured in the arithmetical hierarchy. The arithmetical hierarchy is related to the o-machine based Turing-reducibility via Post's theorem [44]. Therefore it seems a suitable framework for measuring the power of Van Leeuwen and Wiedermann's models, which are related to another model of computation making use of external information resources: interactive Turing machines with advice. The power of hypercomputational models or systems can be measured by identifying the class of sets in the arithmetical hierarchy they can decide, or the class of functions they can compute. In Section 3.3, we extended Ord's [42] assessment of capabilities of hypercomputational models by including Turing machines with advice, as well as Van Leeuwen and Wiedermann's [68, 69] models.

We found that Turing machines with advice are as powerful as o-machines with comparable oracles. Therefore, in general, TM/A's with $\emptyset^{(n)}$ advice can solve Δ^0_{n+1} decision problems. Van Leeuwen and Wiedermann's lineages of cognitive transducers and communities of active cognitive transducers, on the other hand, are shown to be equally powerful as interactive Turing machines with advice. Like interactive Turing machines with advice, the models compute translations over infinite streams rather than over finite strings. Therefore, their power seems not to be measurable with respect to the arithmetical hierarchy — a framework based on functions from Σ^* to Σ^* or subsets of the natural numbers — in a straightforward way. In Section 3.2, we explored the possibility of taking a different view on interactive computation, in order to compare the power of interactive Turing machines relative to an interaction partner to that of o-machines relative to an oracle. In Section 3.3, we conjectured that in this view, the set of streams producible by interactive Turing machines with an interaction partner whose comparable oracle is in $\emptyset^{(n)}$ is a subset of Δ^0_{n+1} streams. We proposed to compare interactive Turing machines with advice to o-machines with two oracles: one comparable to the interaction partner and the other to the advice. We conjectured that the set of streams producible by interactive Turing machines with advice with an interaction partner whose comparable oracle is $\emptyset^{(k)}$ and with an advice whose comparable oracle is $\emptyset^{(l)}$ is a subset of Δ^0_{n+1}, where $n = \max(k, l)$. Since lineages of cognitive transducers are proven to be as powerful as interactive Turing machines with advice, we conjectured that the set of streams producible by lineages of cognitive transducers with $\emptyset^{(k)}$ interaction partner and $\emptyset^{(l)}$ evolution function is a subset of Δ^0_n output streams. We conjectured that the same holds for communities of active cognitive transducers with $\emptyset^{(k)}$ interaction partner and $\emptyset^{(l)}$ description function. The proposal of comparing interaction to relative computation and the conjectures regarding the relative powers of interactive Turing machines, interactive Turing machines with advice, lineages of cognitive transducers and communities of active cognitive transducers require more investigation.

The Wolfram classes [71, 72] of complex system behaviour, which we discussed in Section 4.3, seem a suitable framework for classifying Van Leeuwen and Wiedermann's models, because of the emphasis on behaviour. On the other hand, the Wolfram classes are based on the cellular automaton model, and there are some important differences between the cellular automaton and Van Leeuwen and Wiedermann's communities of active cognitive transducers. First, the components of a cellular automaton have the computing power of finite state machines, whereas the components of communities of active cognitive transducers have the computing power of interactive Turing machines. This is because the individual active cognitive transducers, which are basically only finite state transducers, are able to walk around in their environment and use it as an external memory. Second, communities of active cognitive transducers compute under an interactive scenario, whereas the computations performed by cellular automata resemble those performed by standard Turing machines. The initial configuration is regarded as the input, and during the computation, there is no external input. Cellular automata are thus closed systems, like Turing machines, whereas communities of active cognitive transducers are open systems. Third, the rules governing a standard cellular automaton do not change over time. On the contrary, the composition of communities of active cognitive transducers does change over time. Taking the three differences into consideration, the Wolfram classes seem not to be easily applicable as a framework for classifying the behaviour of Van Leeuwen and Wiedermann's models. Moreover, Wolfram's classification of cellular automaton behaviour is based on empirical study. It would be interesting to study the qualitative behaviour of cellular automata extended with the ability to interact with an external environment, and evolve themselves.

Van Leeuwen and Wiedermann's [68, 69] two main theorems, restated here in Theorem 2.3.1 and Theorem 2.3.2, indeed show that lineages of cognitive transducers and communities of active cognitive transducers are hypercomputational models. Now we discuss what this result means for the computational power of the physical systems they model: complex adaptive systems, including natural and artificial living systems. This discussion is based on our account of physical realisability and exploitability in the context of hypercomputation in Section 3.4.

Physical realisability of hypercomputational models concerns the possibility to construct physical systems that implement the model. The issue of realisability is especially relevant to Van Leeuwen and Wiedermann's claim, because it is a claim about the hypercomputational power of systems which are actually realised in nature. Still, the question remains whether these realisations are capable of hypercomputation like their models. In order to answer this question, one needs to take both the model and the resources used by the model to gain hypercomputational power into account.

Basically, both lineages of cognitive transducers and communities of active cognitive transducers are finite state transducers, and therefore physically realisable. However, the models gain their hypercomputational power by using the resources of infinite operation, interaction, and evolution simultaneously. The use of infinite operation does not pose a problem for the physical realisability of lineages of cognitive transducers and communities of active cognitive transducers. The use of interaction is somewhat more debatable. Interactive machines can produce non-Turing-computable output if they are supplied with non-Turing-producible input from a hypercomputational interaction partner. This makes the resource of interaction quite comparable to external information resources. It shows that in order to produce non-Turing-computable output, physical realisations of interactive machines need to have a physical hypercomputational interaction partner. Such an interaction partner could be a non-Turing-computable quantity, or a non-Turing-computable process, produced by a hypercomputational system. On the other hand, interaction itself need not to happen in a Turing-computable way. Interactive systems can produce non-Turing-computable output by communicating or interacting in a non-Turing-computable way with their interaction partners. This could happen for example if the interactive systems would interact with their interaction partners asynchronously, according to a non-Turing-computable timing function. The use of the resource of evolution is crucial to the hypercomputational power of lineages of cognitive transducers and communities of active cognitive transducers. Together with infinite operation, evolution implicitly yields infinite specification. In order for physical evolving systems to be hypercomputational, their evolution must proceed according to a non-Turing-computable process.

Whether the physical realisations of lineages of cognitive transducers and communities of active cognitive transducers are hypercomputational thus depends on the existence of non-Turing-computable information sources in nature. In Section 3.4, we saw that it is not clear whether the universe actually accommodates non-Turing-computable quantities and processes. It might as well turn out that the universe can in principle be simulated completely by a standard Turing machine. On the other hand, it is also not totally implausible that non-Turing-computable quantities and processes do exist in nature. An answer to this question is up to empirical physics.

Exploitability concerns the question whether the hypercomputational power of hypercomputational systems can be exploited to solve given non-Turing-computable problems.

The issue of exploitability is especially relevant to Van Leeuwen and Wiedermann's claim, because at least in the engineering approach to ALife, artificial living systems are intended to solve given problems. Therefore, we discuss whether the hypercomputational power of the physical counterparts of lineages of cognitive transducers and communities of active cognitive transducers is exploitable. First of all, since both lineages of cognitive transducers and communities of active cognitive transducers rely on infinite operation for their hypercomputational power, their physical counterparts also do. This is problematic, because all that we experience is finite, and it seems implausible that we can observe more than a finite part of the computation in finite time. There are some proposed ways to view an infinite computational process in finite time, such as using Malament-Hogarth space-times, but these are also problematic. Second, the hypercomputational power of lineages of cognitive transducers and communities of active cognitive transducers comes from the use of non-Turing-computable information sources. Even if non-Turing-computable quantities or processes existed in nature, which could act as interaction partners, timing functions, or evolutionary processes for the physical realisations of lineages of cognitive transducers and communities of active cognitive transducers, the question whether the systems could use their hypercomputational power to solve a given problem would remain. First, a non-Turing-computable quantity must be a real value, and accessing such a value would require infinite precision measurement. Moreover, in order to solve a given non-Turing-solvable problem, the quantity or process governing the evolution of the system must be exactly the quantity or process needed by the system to solve the given problem. The problem is, that since the quantity or process is non-Turing-computable, it is not effectively decidable whether the quantity or process at hand is indeed the required one.

We conclude that Van Leeuwen and Wiedermann's lineages of cognitive transducers and communities of active cognitive transducers are basically physically realisable. Although the models are hypercomputational, the question whether their physical counterparts are also hypercomputational is an open question, up to empirical physics. If so, whether the hypercomputational power of these physical systems can be exploited for solving a given non-Turing-solvable problem is not impossible, but seems highly implausible according to current physics. Still, irrespective of the physical realisability and exploitability of their models, Van Leeuwen and Wiedermann's study of computational models of complex adaptive system is very relevant. The models better capture the features of complex adaptive system than Turing machines do, even though it may turn out that their behaviour can in principle by simulated by Turing machines.

Emergence

Finally, we discuss whether Van Leeuwen and Wiedermann's [68, 69] results indeed involve a case of emergence. In Chapter 5, we have seen that the term 'emergence' is used in a variety of contexts, with a variety of different meanings. Implicitly, Van Leeuwen and Wiedermann seem to refer to an emergent property as a property of the whole which is 'more' than the sum of the properties of its constituting parts. In Van Leeuwen and Wiedermann's words, "a community of active cognitive transducers has a much greater computing power than just the 'sum' of the powers of the individual transducers. Here we see the emerging super-recursive computing power" [69, p. 213].

We stress that Van Leeuwen and Wiedermann suggest that it is the hypercomputational power which is an emergent property of their modelled systems. Therefore, we first discuss whether hypercomputational power can be considered an emergent property in the first place. As we have seen in Chapter 5, there are several notions of emergence, each with their own conditions for a property to be emergent. In general, an emergent property is a property of a complex thing that is not a property of its proper parts. Certainly, communities of active cognitive transducers model multi-levelled systems, and the hypercomputational power is a property at the system level, which is not held by any of its individual components. Furthermore, the emergence referred to seems to be a case of synchronic emergence in Humphreys [30] temporal taxonomy, because it involves the simultaneous coexistence of the higher-level property hypercomputational power and lower-level property Turing-equivalence. A problem, on the other hand, is that although computational power is a property of a system, it is not a property that can be associated with having a certain state at a certain time that can be observed. Therefore, as an emergent property, hypercomputational power does not seem to fit in the definitions of emergence we discussed in Chapter 5.

There seems to be another case of emergence involved in Van Leeuwen and Wiedermann's results, one that is specifically relevant because of its relation to ALife. Bedau's [3] notion of weak emergence seems applicable to the behaviour of communities of active cognitive transducers. Weak emergence refers to an emergent property as a property of a system, such that an answer to the question "does the system have property P?" is deducible from the external conditions and the microdynamic, but only by simulation. By Rice's theorem [47], non-trivial properties of the behaviour of Turing machines are not Turing-computable from the Turing machines' description. This holds for communities of active cognitive transducers as well, because they are even more powerful than Turing machines. Moreover, in general, the description function of communities of active cognitive transducers need not be Turing-computable. Therefore, although at each time the configuration of the community is fully constituted out of the configurations of the active cognitive transducers in the community, answers to questions concerning the configuration of a community of active cognitive transducers at a given time can in general not be computed. They can, on the other hand, be obtained by simulating, or running the community of active cognitive transducers. Therefore, the behaviour of communities of active cognitive transducers can be considered weakly emergent. According to Humphreys [30], we may speak of weak emergence as a case of diachronic inferential emergence.

We conclude that although hypercomputational power is a property had by communities of active cognitive transducers while not had by their constituting parts, it does not seem to fit into the definition of an emergent property in the different notions of emergence we discussed in Chapter 5. Rather than the hypercomputational power of communities of active cognitive transducers, it is the behaviour of communities that is emergent. More specifically, it is weakly emergent. This case of emergence is especially relevant, because weak emergence is ubiquitous in complex systems, and communities of active cognitive transducers are models of such systems.

6.2 Implications

In this section, we point to a number of implications of Van Leeuwen and Wiedermann's results for the field of ALife, based on our discussion of ALife in Chapter 4. Although Van Leeuwen and Wiedermann's work in [68, 69] is situated in an ongoing debate in the theory of computation, focussed on new computational paradigms, the authors' results do not seem to have made an entry in the field of ALife. This is surprising, since Van Leeuwen and Wiedermann's conclusion that artificial living systems are more powerful than often believed could have major implications for ALife.

Van Leeuwen and Wiedermann's results seem to have the most immediate implications for soft ALife. This is because in soft ALife, the digital computer is used as a tool for synthesising lifelike behaviour, and Van Leeuwen and Wiedermann's results are computability theoretic ones. If Van Leeuwen and Wiedermann's claim is true, living systems may show behaviours which cannot be generated by a standard Turing machine. This stresses the potential limitations of general purpose computers as tools for synthesising life-like behaviour. That is, general purpose computers computing in isolation as a closed system, without for example access to the Internet, a true random generator, or unpredictable user input.

At the same time, Van Leeuwen and Wiedermann's models and the non-uniform interactive paradigm they are part of, point to some directions for overcoming the potential limitations of computers as tools for synthesising life. The three features of the non-uniform interactive paradigm are infinite operation, interaction, and non-uniform evolution. Infinite operation is already reflected in the computational view of soft ALife, whose emphasis is on ongoing behaviour rather than final results. Moreover, the emphasis in soft ALife is already on networks of parallelly operating and interacting simpler programs, devices or systems. In parallel computation, asynchrony becomes an issue. Van Leeuwen and Wiedermann's results suggest that complex adaptive system behaviour can in general only be generated if the systems also evolve during the simulation. Interaction, asynchrony and evolution allow for the influence of possibly non-Turing-computable information on the behaviour of complex adaptive systems. Natural living systems are situated in a real word environment, which may inhibit non-Turing-computable processes. Therefore, the key to generating life-like behaviour in an artificial way seems to be making use of exogenous information. Environments produced in an algorithmic way seem not to be good candidates for supplying such information, because they are necessarily Turing-computable. Situating artificial systems in a real world environment, already commonplace in hard ALife, seems to be a good solution. Situated agents can exploit their complex environment to generate possibly richer behaviour.

Although letting possibly non-Turing-computable information enter artificial living systems via openness to a real environment leads to richer behaviour, it also leads to an increase in unpredictability of the systems' behaviour. This may pose a serious problem for the engineering approach to ALife, which is aimed at designing systems for solving given complex problems. Van Leeuwen and Wiedermann's results suggest that non-Turing-computable inputs from the environment may be exploited for solving even non-Turing-solvable problems. However, as we saw in Section 3.4, these external inputs must be exactly right for the problem at hand. This means that care must be taken to ensure that the environment is 'cooperative'. However, this is problematic, since there is no effective way to check whether the environment indeed behaves as desired.

In conclusion, there seems to be a trade-off for ALife. On the one hand, it may be that ALife can only succeed in generating life-like behaviour through artificial systems by letting possibly non-Turing-computable information influence these systems. On the other hand, opening systems up to such information may make it extremely hard, or even impossible to retain control over the behaviour of artificial living systems.

Bibliography

[1] Alexander, S. *Space, time, and deity.* Macmillan, London, 1920.

[2] Balcazar, J., Diaz, J., and Gabarro, J. *Structural Complexity I.* Texts in Theoretical Computer Science. An EATCS Series. Springer, Berlin, 1995.

[3] Bedau, M. A. Weak emergence. *Noûs 31* (1997), 375–399. Supplement: Philosophical perspectives 11: Mind, causation, and world.

[4] Bedau, M. A. Artificial life: organization, adaptation and complexity from the bottom up. *TRENDS in Cognitive Sciences 7*, 11 (November 2003), 505–512.

[5] Bedau, M. A. Artificial life. In *Philosophy of Biology*, M. Matthen and C. Stephens, Eds., Handbook of the Philosophy of Science. North-Holland, Amsterdam, The Netherlands, 2007, pp. 595–613.

[6] Bonabeau, E. W., and Theraulaz, G. Why do we need artificial life? In *Artificial life: An overview*, C. G. Langton, Ed., second printing ed. The MIT Press, Cambridge, Massachusetts, 1996, pp. 303–325.

[7] Broad, C. *The mind and its place in nature.* Routledge & Kegan Paul, London, 1925.

[8] Brooks, R. A. A robot that walks: Emergent behaviours from a carefully evolved network. *Neural Computation 1*, 2 (Summer 1989), 253–262.

[9] Church, A. A set of postulates for the foundation of logic. *Annals of Mathematics 34*, 4 (October 1933), 839–864.

[10] Church, A. An unsolvable problem of elementary number theory. *American Journal or Mathematics 58*, 2 (April 1936), 345–363.

[11] Cook, M. Universality in elementary cellular automata. *Complex systems 15*, 1 (2004), 1–40.

[12] Cooper, S. B. *Computability Theory.* Chapman & Hall, Boca Raton, 2004.

[13] Cooper, S. B. Computability and emergence. In *Mathematical Problems from Applied Logics: New Logics for the XXIst Century*, D. M. Gabbay, S. S. Goncharov, and M. Zakharyaschev, Eds., vol. 4 of *International Mathematical Series.* Springer, New York, 2006, pp. 193–232.

[14] Cooper, S. B. Emergence as a computability-theoretic phenomenon. *Applied Mathematics and Computation 215*, 4 (October 2009), 1351–1360.

[15] Cooper, S. B., and Odifreddi, P. Incomputability in nature. In *Computability and models*, S. B. Cooper and S. S. Goncharov, Eds., The University series in mathematics. Kluwer Academic/Plenum Publishers, New York, USA, 2003, pp. 137–160.

[16] Copeland, B. J. Even Turing machines can compute uncomputable functions. In *Unconventional models of computation*, C. S. Calude, J. Casti, and M. J. Dinneen, Eds. Springer, Singapore, 1998, pp. 150–164.

[17] Copeland, B. J. Hypercomputation: philosophical issues. *Theoretical Computer Science 317*, 1–3 (June 2004), 251–267.

[18] Copeland, B. J., and Proudfoot, D. Alan Turing's forgotten ideas in computer science. *Scientific American 280*, 4 (April 1999), 98–103.

[19] Copeland, B. J., and Sylvan, R. Beyond the universal Turing machine. *Australian Journal of Philosophy 77*, 1 (March 1999), 46–67.

[20] Davies, E. B. Building infinite machines. *British Journal for Philosophy of Science 52*, 4 (2001), 671–682.

[21] Davis, M. The myth of hypercomputation. In *Alan Turing: Life and legacy of a great thinker*.

[22] Davis, M. Why there is no such discipline as hypercomputation. *Applied Mathematics and Computation 178*, 1 (July 2006), 4–7.

[23] Gödel, K. On undecidable propositions of formal mathematical systems. In *The undecidable*, M. Davis, Ed. Raven Press Books, Hewlett, 1965, pp. 41–74. Notes by S. C. Kleene and J. B. Rosser on lectures at Princeton.

[24] Goldin, D., and Wegner, P. The Church-Turing thesis: Breaking the myth. In *New computational Paradigms* (Berlin, 2005), S. Cooper, B. Löwe, and L. Torenvliet, Eds., vol. 3526 of *Lecture Notes in Computer Science*, CiE 2005, Springer, pp. 152–168.

[25] Hamkins, J. D., and Lewis, A. Infinite time Turing machines. *The Journal of Symbolic Logic 65*, 2 (June 2000), 567–604.

[26] Hartley Rogers, J. *Theory of recursive functions and effective computability.* McGraw-Hill series in higher mathematics. McGraw-Hill, New York, 1967.

[27] Hogarth, M. L. Does general relativity allow an observer to view an eternity in a finite time? *Foundations of Physics letters 5*, 2 (April 1992), 173–181.

[28] Holland, J. H. *Adaptation in natural and artificial systems.* University of Michigan Press, 1975.

[29] Holland, J. H. Studying complex adaptive systems. *Journal of Systems Science and Complexity 19*, 1 (2006), 1–8.

[30] Humphreys, P. Computational and conceptual emergence. *Philosophy of Science 75* (December 2008), 584–594.

[31] Karp, R. M., and Lipton, R. J. Some connections between non-uniform and uniform complexity classes. In *Proceedings of the twelfth annual ACM Symposium on the Theory of Computing* (New York, 1980), R. E. Miller, S. Ginsburg, W. A. Burkhard, and R. J. Lipton, Eds., ACM, pp. 302–309.

[32] Kim, J. Making sense of emergence. *Philosophical Studies 95*, 1-2 (August 1999), 3–36.

[33] Kim, J. Emergence: Core ideas and issues. *Synthese 151*, 3 (August 2006), 547–559.

[34] Langton, C. G. Artificial life. In *Artificial life* (Redwood City, 1989), C. G. Langton, Ed., vol. VI of *Santa Fe Institute studies in the sciences of complexity*, Addison-Wesley, pp. 1–48.

[35] Langton, C. G. Computation at the edge of chaos: Phase transitions and emergent computation. *Physica D 42*, 1-3 (June 1990), 12–37.

[36] Langton, C. G. Editor's introduction. In *Artificial life: An overview*, C. G. Langton, Ed., second printing ed. The MIT Press, Cambridge, Massachusetts, 1996, pp. ix–xi.

[37] Maes, P. Modeling adaptive autonomous agents. In *Artificial life: An overview*, C. G. Langton, Ed., second printing ed. The MIT Press, Cambridge, Massachusetts, 1996, pp. 135–162.

[38] Mill, J. S. *A system of logic.* Longmans, Green, Reader, and Dyer, London, 1843.

[39] Milner, R. Elements of interaction. *Communications of the ACM 36*, 1 (January 1993), 78–89.

[40] Morgan, C. *Emergent evolution.* Williams and Norgate, London, 1923.

[41] O'Connor, T., and Wong, H. Y. Emergent properties. In *The Stanford Encyclopedia of Philosophy*, E. N. Zalta, Ed., spring 2009 ed. 2009.

[42] Ord, T. *Hypercomputation: Computing more than the Turing machine.* PhD thesis, University of Melbourne, 2002.

[43] Ord, T. The many forms of hypercomputation. *Applied Mathematics and Computation 178*, 1 (July 2006), 143–153.

[44] Post, E. L. Degrees of unsolvability. *Bulletin of the American Mathematica Society 54*, 7 (1948), 241–242.

[45] Prasse, M., and Rittgen, P. Why Church's thesis still holds. Some notes on Peter Wegner's tracts on interaction and computability. *The Computer Journal 41*, 6 (June 1998), 357–362.

[46] Putnam, H. Reductionism and the nature of psychology. *Cognition 2*, 1 (1973), 131–146.

[47] Rice, H. G. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society 74*, 2 (March 1953), 358–366.

[48] Ronald, E. M. A., Sipper, M., and Capcarrère, M. S. Design, observation, surprise! A test of emergence. *Artificial Life 5*, 3 (Summer 1999), 225–239.

[49] Schoenfinkel, M. Über die Bausteine der mathematischen Logik. *Mathematische Annalen 92*, 3-4 (September 1924), 305–316.

[50] Schröder, J. Emergence: Non-deducibility or downward causation? *The Philosophical Quarterly 48*, 193 (1998), 433–452.

[51] Soare, R. I. The history and concept of computability. In *Handbook of Computability Theory*, E. R. Griffor, Ed., vol. 140 of *Studies in Logic and the Foundations of Mathematics*. Elsevier Science, Amsterdam, 1999.

[52] Stannett, M. Computation and hypercomputation. *Minds and machines 13*, 1 (February 2003), 115–153.

[53] Stannett, M. The case for hypercomputation. *Applied Mathematics and Computation 178*, 1 (July 2006), 8–24.

[54] Taylor, C., and Jefferson, D. Artificial life as a tool for biological inquiry. In *Artificial life: An overview*, C. G. Langton, Ed., second printing ed. The MIT Press, Cambridge, Massachusetts, 1996, pp. 1–14.

[55] Thomas, W. Automata on infinite objects. In *Formal models and semantics*, J. van Leeuwen, Ed., vol. B of *Handbook of theoretical computer science*. Elsevier, Amsterdam, 1990, pp. 133–191.

[56] Turing, A. M. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society s2-42*, 1 (1937), 230–265.

[57] Turing, A. M. Systems of logic based on the ordinals. *Proceedings of the London Mathematical Society s2-45*, 1 (1939), 161–228.

[58] van Leeuwen, J. Conversation, 23 November 2011.

[59] van Leeuwen, J. Conversation, September 2011.

[60] van Leeuwen, J., and Wiedermann, J. On algorithms and interaction. In *Mathematical Foundations of Computer Science* (Berlin, 2000), M. Nielsen and B. Rovan, Eds., no. 1893 in Lecture Notes in Computer Science, MFCS 2000, Springer, pp. 99–112.

[61] van Leeuwen, J., and Wiedermann, J. The Turing machine paradigm in contemporary computing. In *Mathematics Unlimited - 2001 and Beyond* (Berlin, 2000), B. Engquist and W. Schmid, Eds., Springer, pp. 1139–1155.

[62] van Leeuwen, J., and Wiedermann, J. Beyond the Turing limit: Evolving interactive systems. In *SOFSEM 2001: Theory and practice of informatics* (Berlin, 2001), L. Pacholski and P. Ruzicka, Eds., vol. 2234 of *Lecture Notes in Computer Science*, SOFSEM 2001, Springer, pp. 90–109.

[63] van Leeuwen, J., and Wiedermann, J. A theory of interactive computation. In *Interactive computation: The new paradigm*, D. Goldin, S. A. Smolka, and P. Wegner, Eds. Springer, Berlin, 2006, pp. 119–142.

[64] Verbaan, P. *The computational complexity of evolving systems.* PhD thesis, Utrecht University, 2005.

[65] von Neumann, J. *Theory of self-reproducing automata.* University of Illinois Press, Urbana, 1966. Edited and completed by A. W. Burks.

[66] Wegner, P. Why interaction is more powerful than algorithms. *Communications of the ACM 40*, 5 (May 1997), 80–91.

[67] Weihrauch, K. *Computable analysis: An introduction.* Texts in Theoretical Computer Science: An EATCS Series. Springer, Berlin, 2000.

[68] Wiedermann, J., and van Leeuwen, J. Emergence of a super-Turing computational potential in artificial living systems (extended abstract). In *Advances in artificial life* (Berlin, 2001), J. Kelemen and P. Sosík, Eds., vol. 2159 of *Lecture Notes in Computer Science*, ECAL 2001, Springer, pp. 55–65.

[69] Wiedermann, J., and van Leeuwen, J. The emergent computational potential of evolving artificial living systems. *AI Communications 15*, 4 (2002), 205–215.

[70] Wiedermann, J., and van Leeuwen, J. How we think of computing today. In *Logic and theory of algorithms* (Berlin, 2008), A. Beckmann, C. Dimitracopoulos, and B. Löwe, Eds., vol. 5028 of *Lecture Notes in Computer Science*, CiE 2008, Springer, pp. 579–593.

[71] Wolfram, S. Cellular automata as models of complexity. *Nature 311* (October 1984), 419–424.

[72] Wolfram, S. Universality and complexity in cellular automata. *Physica 10D 10*, 1-2 (January 1984), 1–35.

Index