Doctoral Thesis

Explicit design, and adaptation in self-construction

Author(s): Roth, Fabian

Publication Date: 2007

Permanent Link: https://doi.org/10.3929/ethz-a-005328389

Rights / License: In Copyright - Non-Commercial Use Permitted
Explicit Design, and Adaptation in Self-Construction

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY (ETH)
ZÜRICH

for the degree of
DOCTOR OF NATURAL SCIENCES

presented by
FABIAN ROTH
Dipl. Math. ETH
born July 25, 1975
citizen of Zürich, Switzerland

accepted on the recommendation of
Prof. Dr. Rodney J. Douglas, examiner
Prof. Dr. Hava T. Siegelmann, co-examiner
Prof. Dr. Christoph von der Malsburg, co-examiner

2007
Acknowledgements

I want to thank Rodney Douglas, my thesis advisor and inspirational source very much for his great support. His guidance was very gentle and he was very careful not to disturb my own intuition and approach to the field while providing guidelines and stopping me when I was getting too far off track. Rodney’s support went well beyond scientific supervision: He helped me gain self-confidence about my project and motivated me in difficult times. It was a great privilege for me to work on such a visionary project. A project that might not have found many homes at the current stage of technology – but the stimulating environment created by the professors Kevan Martin and Rodney Douglas in the Institute of Neuroinformatics have provided the perfect balance between pioneering spirit and grounding to reality.

I would also like to thank Hava Siegelmann for accompanying me along the first part of my thesis and for the inspiring discussions we held. I am very happy that she and Christoph von der Malsburg can be the coexaminers of my thesis and I would like to thank them both very much for going through the effort of reading my thesis and travelling to Switzerland for this purpose.

We had many and thorough discussions about my work in Rodney’s group Rodney and Friends with its members Jörg Conradt, Dylan Muir, Andreas Steimer, Frédéric Zubler, Bobby Rohrkemper and Daniel Rubin. I am greatful for the time they have all invested and the useful comments they have provided. Matthew Cook, a more recent member of INI, has helped me with the analysis of the behavior of a Braitenberg vehicle (as presented in Appendix A). I also would like to thank my office members Reto Wyss, Andreas Bäbler, Pamela Baker, Armin Duff and Alexis Guanella for the time we had in our séparée downstairs.

Very special thanks go to Nadja, my wife. She has offered me great trust, love and encouragement that helped me to keep going from the very beginning to the end of this work. I also owe great thanks to my parents and my brother, for always supporting me and being the family that lead me to the point I am now.
Abstract

Man-made systems are becoming increasingly complex and autonomous. However, traditional engineering methods appear to be reaching the limit of design and construction of such complex systems, and so, novel methods must be developed. Consequently, much scientific effort is now being directed towards understanding self-constructing systems, because they promise many advantages over traditional direct feed-forward methods. For example: (1) Self-construction utilizes simple but incremental rules and local construction processes to produce complex phenotypes, (2) self-constructing systems can continuously adapt their structure to their environment and (3) self-repair is a direct consequence of self-construction. At present, however, these exciting properties lack any substantial theoretical and practical understanding. One of the prominent intellectual challenges is that, unlike conventionally constructed systems, self-constructing systems cannot rely on a global observer, nor on a random access construction arm. Instead, the global coherence of the construction process must arise out of local co-operative processes both for observation and construction. There is as yet little formal understanding of how to achieve such global coherence in a design-oriented manner.

Previous studies of self-construction have emphasized evolutionary approaches to find appropriate self-construction instructions. However, these methods do not give us insights about useful mechanisms of self-construction because the true mechanisms are obscured by many other non-functional but distracting genes. Furthermore, trial and error searches such as genetic algorithms will not be feasible anymore as systems are becoming larger and more complex because they take too long to converge. By contrast, in this thesis, we propose how to design self-constructing and self-repairing systems by explicit hierarchical specifications.

The system we investigate develops from a single precursor cell, and grows through self-replication into a predesigned functional multicellular system. As in biological systems, every cell carries the same description of
the final system similar to a genetic code. When the cell replicates, this description is cloned into an identical copy and placed into the daughter cell. The interplay between individual cells and the gradually increasing self-created complexity of the local structure that surrounds them causes the serial unfolding of the final functional organism. The developed structure feeds back continuously to the development process, and so the final state of the system is its equilibrium state. Through this paradigm, the system is capable of robust construction and self-repair.

We demonstrate how a simple model multicellular organism can assemble itself by replication from a single cell, and finally express a fundamental behavior: foraging. Furthermore, we investigate mechanisms that will allow a self-constructing system to learn from the environment and adapt throughout its development phase. This decentralized learning process in the cells is very similar to multi-agent learning and so faces similar pitfalls. One of the most dominant problem is credit assignment: When many cells act together to yield a single global behavior, it is very difficult for a single cell to interpret its effect with respect to the global reward. Another problem that arises is the instability in the global behavior caused by the incoherent structure and function of the still learning cells. This instability in behavior leads to an incoherent reward signal that can not be used for learning. Here, we propose how to resolve these problems by reduction of the degrees of freedom through self-organization. We demonstrate successful adaptation to two example environmental tasks.

In the second part of this thesis we extend the principles developed in the first part to more general networks and communicating processes and propose a scheme for communication between processes embedded in a network of unknown topology. We present mechanisms that allow single processes to grow into predefined networks of communicating processes and maintain themselves. Finally, we demonstrate the self-construction of structures that reflect the causal dynamics of the world they are observing through sensory processes. Because these networks of processes build a model of the world, they are capable of prediction. We show how the predictive power of such a network can be used to detect novelty in the world.
Zusammenfassung


Wir untersuchen ein System das aus einer einzelnen Zelle heraus


Contents

1 Introduction 1
  1.1 Engineered versus Biological Construction .............. 2
  1.2 Learning in Construction .................................. 6
  1.3 Past Literature .............................................. 8

2 Complexity, Machines and Organisms 13
  2.1 Complexity .................................................. 13
    2.1.1 Measures of Complexity ................................. 14
    2.1.2 Complexity depends on the Interpretation ............. 15
    2.1.3 How does Complexity emerge? ........................... 18
  2.2 Machines and Organisms .................................... 21
    2.2.1 Engineered Modules ..................................... 22
    2.2.2 Natural Modules .......................................... 23
    2.2.3 Differences in Organization ............................. 24
  2.3 The Limits of Engineered Design and Construction ....... 25
    2.3.1 Too complex for human comprehension ................. 26
    2.3.2 High level languages and Compilers can help .......... 27
    2.3.3 Self-Construction requires novel Design principles ... 28
  2.4 Conclusion .................................................. 29

3 Self-Construction 31
  3.1 Von Neumann’s theory of self-replicating systems .......... 32
  3.2 Construction with Feedback .................................. 35
    3.2.1 Traditional feed-forward construction .................. 36
    3.2.2 Implicit description without environmental input ...... 37
    3.2.3 Implicit description with environmental input ........... 37
    3.2.4 Four feedback loops that control development .......... 39
  3.3 A cybernetic view on self-construction ...................... 41
    3.3.1 Construction ............................................. 43
    3.3.2 Symbolizing and Learning ................................ 45
  3.4 Conclusion .................................................. 46
# CONTENTS

4 A Self-Constructing Artificial Organism 47
  4.1 Introduction .................................................. 47
  4.2 The Model ................................................... 50
    4.2.1 The Cell ................................................. 50
    4.2.2 The Environments ....................................... 53
  4.3 The modular factories ....................................... 54
    4.3.1 Reactor factories ...................................... 55
    4.3.2 Competence factories .................................. 57
  4.4 Design and Development ..................................... 60
    4.4.1 Design of the Code .................................... 60
    4.4.2 The Description Code .................................. 62
    4.4.3 Development ........................................... 64
  4.5 Behavior and Self-repair .................................... 66
  4.6 Conclusion .................................................. 71

5 Learning in Development 75
  5.1 Introduction .................................................. 75
  5.2 A Local Structural Learning Rule ............................ 77
  5.3 Caveats of Multiagent Learning ............................... 78
  5.4 Local Reward Modulation ..................................... 80
    5.4.1 Simulation .............................................. 81
    5.4.2 Organism ................................................. 82
  5.5 Reduction of Degrees of Freedom ............................. 83
  5.6 Conclusion .................................................. 85

6 Networks of Processes 87
  6.1 Introduction .................................................. 87
  6.2 The Processes and Messages ................................... 88
  6.3 Dynamic Routing ............................................. 89
    6.3.1 Previous Work .......................................... 90
    6.3.2 Age Based Q-Routing ................................... 91
    6.3.3 Results ................................................ 93
  6.4 Process Migration ............................................ 95
  6.5 Graph Construction .......................................... 100
  6.6 Connectivity from Observation ............................... 104
    6.6.1 A Causal Learning Rule .................................. 107
    6.6.2 Analysis of the Learning rule ......................... 110
    6.6.3 Simulation in Network of Processes .................... 111
  6.7 Connectivity from Temporal Correlation ...................... 115
    6.7.1 Prediction and Novelty .................................. 117
  6.8 Conclusion .................................................. 118
Chapter 1

Introduction

A few years ago, during the intercontinental race for the first touchdown on Mars, the mission of the European Mars surface explorer Beagle 2 failed due to unknown reasons. In the FAQ on their official website[4] ESA states:

Mars Express reached Mars at the end of December 2003. Six days before entering into orbit around Mars, Mars Express ejected the Beagle 2 lander. The orbiter was inserted into orbit around Mars on 25 December 2003. The Beagle 2 lander failed to make radio contact shortly after it was due to land on Mars on Christmas Day. Various attempts were made to contact the lander. However, the Beagle 2 Management Board met in London on 6 February and, following an assessment of the situation, declared Beagle 2 lost.

Hypotheses for its failure range from simple antenna disruptions to scenarios where Beagle 2 never even reached the Mars surface. Most likely, it was physical damage that caused the silence. And there was no one there to see the problem. And worse yet, no one there to fix it. Future missions could work out more sophisticated contingency plans that cover more possible but unlikely events and their resolutions and so lower the overall chance of mission failure. However, the core of the problem would thereby not be solved: An unpredictable event that causes an undefined state of the system can not be resolved autonomously. To solve this problem a system would have to be able to sense its own state (physical as well as internal) and include a means to change its configuration if necessary, for example by changing its physical structure, i.e. repairing itself.

More recently, at the time this thesis is written, the serialized construction of the Airbus 370, one of the most complex systems of its time,
has been delayed by months due to \textit{wiring problems}, thus causing an immense deficit for the company Airbus. Airbus officially communicated to the press:\cite{2}:

In June [2006], the amount of work to be done to finalise the installation of the electrical harnesses into the forward and rear section of the fuselage had been underestimated. Beyond the complexity of the cable installation, the root cause of the problem is the fact that the \textit{3D} Digital Mock up, which facilitates the design of the electrical harnesses installation, was implemented late and that the people working on it were in their learning curve.

In other words: The complexity of the design broke the manufacturing process. Even though the structure is exactly specified, the way how to assemble it is non-trivial to extract from the explicit specification. It is often seen in construction plants that additionally to the specified assembling instructions, a great deal of \textit{non-official} assembling instructions are collected and utilized to achieve the construction. The fact that problems of assembly are non-predictable at the time of the design process discloses a fundamental flaw in current design and engineering: Treating these processes as two consequent stages makes construction of complex systems unfeasible.

These are just two examples that illustrate where modern engineering is currently reaching its limit. It is a hard limit that can not simply be extended by faster computers or better hardware or more refined construction mechanisms. Extending beyond it requires novel methods of engineering. Both mentioned deficiencies can be resolved by a single new concept: Self-Construction.

Nature successfully employs \textit{self-construction} by multicellular development to tackle these kinds of problems. Following the biological approach, in this thesis we will thus explore the question:

Given a self-replicating unit, what are the principles of engineered multicellular self-construction.

1.1 Engineered versus Biological Construction

Figure 1.1 shows two approaches to construction: traditional engineered construction and biological construction. Traditional engineering is essentially a feed forward process: It consists of three phases: Design, Construction and Operation. In the Design phase, an explicit, abstract Description
Engineered Construction

Design | Construction | Operation

Specification → Factory → Instance

Environmental

Biological Construction

Design | Construction | Operation

Specification → Factory → Instance

Environmental

Figure 1.1: Traditional engineered vs biological construction.

is specified. In the Construction phase, the Description is translated into a target physical instance through exterior assembly by a Factory to conform to its blueprint. In the Operation phase, the instance is switched on and confronted with its environment to perform its task. It is only then, when it can be extensively tested. This is usually done in the prototyping stage. The system is trialed carefully and its design is adapted accordingly before it is being sent back into construction. This feedback loop is a very long one and it involves the interaction of a human observer and corrector.

In the production stage all problems need to have been resolved and the system has to perform its function regardless of all disturbances from the environment within the specified range of operating conditions. Environmental influences have to be intercepted by the final structure by a clever design as presented in the specification. However, it is difficult and often impossible to specify the full range of configurations that the target system may experience during operation. Some of these unknown states may lead to critical failures. These are challenges we confront for example in manufacturing and foraging in far away [42] or nano-scale [38, 33, 34, 79, 128] environments.

Nature offers a different approach to construction (Figure 1.1 bottom):
It encapsulates the Description and the Factory inside the target Instance and so constructs from the inside out. By doing this, the Construction and Operation are much closer interleaved and allow for quick adaptation to environmental influences. Local structures can be built on demand. Also, trial and error cycles can be made much shorter. The final organism is very robust against perturbation through the constant balance between construction and operation. Given the complexity of the system, there are astonishingly few defects in construction or failures of performance, despite large variations in operating conditions. Moreover, the organism is able to compensate for injury and infection by self-repair.

Nature builds extremely complex organisms by autonomous construction from a single precursor cell. A simple comparison shows that the human DNA (∼ 30\textsuperscript{‘}000 protein encoding genes[90]) can not possibly store explicit information about the location and type of every single cell in our body (∼ 10\textsuperscript{13} cells) or the location and type of every synapse in the neocortex (∼ 10\textsuperscript{14}[87]). Hence, Nature must employ an indirect encoding that it uses to incrementally construct its target instance rather than an explicit blueprint. Chapters 2 and 3 further discuss direct versus indirect encoding. To name some advantages upfront: An indirect rule-based mapping naturally compresses information about the explicit structure, as is seen in algorithmic information theory (see Chapter 2), where the specification of a (algorithmically) non-random structure can be compressed into an algorithm [24]. Indirect encoding and iterative construction also promotes the automatic emergence of complexity through iterative application of simple rules[126], and the ability to adapt to environmental signals[69].

A first step of the biological style inside-out construction requires a self-replicating unit. In his pioneering work, von Neumann[119] proposed minimal requirements for self-replication: A universal constructor, and an explicit description of the target system. The Constructor reads the description, and translates that information into the construction of the target which in turn is also capable of self-replication. Von Neumann used an explicit one-to-one mapping between the description (a blueprint) and the target system. While his work is a remarkable theoretical achievement, systems of his kind remain subject to the same problems that traditionally engineered systems have: They are not robust against structural defects. Their construction process is purely feed-forward and so insensitive to their own structure during assembly. Hence, they are unable to compensate for deficiencies and perturbations. Moreover, unlike the von Neumann concept, Nature does not encode the entire target system explicitly in the description.

Rather, Nature provides the boundary conditions for development and
Figure 1.2: Waddington's epigenetic landscape[120]. Original captions: a “Part of an epigenetic landscape. The path followed by the ball, as it rolls down towards the spectator, corresponds to the developmental history of a particular part of the egg.” b “The complex system of interactions underlying the epigenetic landscape, which slopes down from above one’s head towards the distance, is controlled by the pull of these numerous guyropes which are ultimately anchored to the genes.”

let the system grow along the attractor of the so created dynamical system. Figure 1.2 by Waddington[120] very nicely illustrates this interpretation in an epigenetic landscape which is an allegory of development as a dynamical system. The genes merely define a landscape, but the actual development is carried out by the physics of the world in a deterministic manner, pictured as a ball running down the landscape. Waddington further states that through a phenomenon called canalization the development of an organism is remarkably robust against both, environmental as well as genetic fluctuations[121]. This also has its allegoric reference in the figure: the ball can only be pushed out of its groove by a large disturbing force.

A problem about the design of implicit descriptions arises because the mapping between the final structure and the description is non-trivial, maybe even non-computable. Stated in terms of Waddington’s allegory this question equates to finding the right genes that will produce the landscape that leads to the wanted final equilibrium structure. After all, evolution took billions of years to optimize its descriptions.

To facilitate design, we will utilize principles of stigmergy[51, 18] by which the rules are defined incrementally depending on the previously built structure and thus define the development as feedback between the construction in progress and its intermediate structure. Thus allowing the developmental process to be designed as a recipe that is piling up modularized construction blocks, rather than an enormous dynamical system with
incomprehensible interdependencies.

D’Arcy Thompson investigated morphogenesis in his book “On Growth and form” [112] in 1917 and realized the importance of feedback in the growth of organism in Nature. In “The Chemical Basis of Morphogenesis” Turing [114] considered interactions of cells in a field theory of chemicals in reaction-diffusion systems. These models are extremely important because they exhibit an essential ingredient of inside-out construction: The ability to generate non-trivial global structures from (maybe identical) local processes. In our model we will use reaction-diffusion systems proposed later by Gierer and Meinhard [47] to create global patterns and form modularized construction blocks that self-organize into their structure.

1.2 Learning in Construction

Learning is the retrieval and integration of information that is not a priori contained in the prespecified description of a system. Biological systems employ learning on many different levels: From structural dependence throughout development to changes in synaptic plasticity. In living systems the most prominent instance of learning is the plasticity in the nervous system: Many of the behavioral traits of living systems are hard-wired in reflexes (low-level) and instincts (high-level) and thus need to be prespecified. However, most of the behavioral traits are learned from the environment and stored in the synapses of the nervous system. The degree to which learning in this fashion takes place varies largely between different species. Simple organisms that are capable only of a small set of interactions with the environment can work sufficiently well as simple reflex machines that don’t need to incorporate any prior experience in their actions. Mammals on the other hand, acquire most of their behavioral function through learning.

In biology there is a trade-off between prespecification and learning. The two exponents of this trade-off are complete prespecification and complete environmentally derived structure and function. Complete prespecification is rigid. If the system is going to have to survive in a variable environment, all possible configurations to tackle the environment need to be contained a priori in the prespecification. As discussed for traditional engineered construction this is often not feasible. On the other hand, in complete environmentally derived structure and function, only a behavioral objective might be encoded in the prespecification. It is then the task of the development process to find the structural implementation that optimizes the objective. However, this is also not feasible because finding the right
1.2. Learning in Construction

Figure 1.3: The development of an example organism. Early in development, the system bootstraps its structure by self-constructing according to rules specified in a Pre-Specification. Later in development, Learning and Adaptation will become more important. Finally, the system develops into a sensorimotor organization that is capable of behavior (similar to Braitenberg vehicle[21])

structure that fulfills the objective of the prespecification is a non-trivial task that often involves trial-and-error in a very high dimensional space in which only a null-set of configurations are acceptable. In Chapter 5 we will look at this problem in more detail.

Evolution found for Nature’s species the perfect balance of prespecification and learning. While more potential for learning and less prespecification would be beneficiary for more optimal adaptation to an unknown and changing environment and optimization of behavior therein, it consumes much more time in development. To optimize survivability, Nature’s species had to balance this feature against a quick hard-wired solution that also does the job in most of the situations to optimize survivability. Some animals need to be able to cope autonomously with their environment within minutes of their birth. Species that have a safer environment to grow up in can allow much more learning throughout their development. This could also explain the fact that the size of the genetic material is similar across a wide variety of different species while their behavior ranges from simple to very complex. Simple behavior that is hardwired can require the same amount of description code as nonspecific information that serves as framework for the learning of more complex behavior.

In this thesis we propose a scheme whereby a primary structure is boot-
strapped from a prespecified description. This primary structure reduces the dimensionality of the search problem in the learning process and allows for ad-hoc adaptation to the environment of the developing organism. In Chapter 5 we experiment with learning in the organism presented in Chapter 4 as illustrated in Figure 1.3. In Chapter 6 we explore the feasibility of extracting an entire model of the world from information acquired from the environment.

1.3 Past Literature

Besides the work refining von Neumann’s approach to simpler cellular systems with fewer states by Arbib[9] and others there has also been more abstract work on self-construction and self-maintenance. Fontana and Buss describe in their work[41] the first steps in their program to develop a formal understanding of self-maintaining organizations. They realize that the traditional theory of dynamical systems is not equipped for dealing with constructive processes. Rather, there should be something like a calculus of objects, a theory of constructive dynamical systems. In their attempt to model constructive systems they use artificial chemistries and realize a very important point: If a formalism is going to model chemistry, it must allow a symbol to be both manipulable and functional at the same time; just like proteins have a structure in 3D-space which is deformable and, at the same time have a catalytic action on chemical processes in the neighborhood. The one existing formalism allowing that is $\lambda$-calculus. This theory might be interesting to investigate principle barriers and qualities of self-maintaining systems, but is too abstract to allow the construction of functioning organizations.

Others have attempted to capture the nature of the living abstractly, such as the Metabolism-Repair systems of Rosen[98] and Casti[23] or Eigen’s[36] theory of Hypercycles reflect the importance of a dynamic systems approach in biology and advise us not to think too algorithmically about Nature.

The following authors have understood this importance and have implemented models: Willshaw and von der Malsburg have concretely exemplified how structure can be guided by function in a process of self-organization[125, 118] in the setup of ordered retinotectical projections. On the cellular level Ono and Ikegami[86] present a stochastic particle model driven by a simple chemical network and demonstrate cell formation, maintenance and division. Nice about this model is that all its properties are emergent from one single set of equations of reactions. Qualita-
tively different effects (like cell maintenance and division) didn’t have to be modeled by qualitatively different rules. Although this model might help understanding the emergence and maintenance of cells, it is too low level for our purposes: A cell formed by this model is merely an organization of chemicals and is not a priori an object. However, entities being treatable as objects allows tackling the problem of network self-assembly from a more abstract and simpler point of view. Again, a theory of constructive dynamical systems as proposed by Fontana and Buss in [41], filling the gap between artificial chemistries and abstract objects would help.

Mjolsness et al.[82] use a connectionist approach to model biological development. An attractor network determines the genetic regulatory and expression equilibrium of the cells while discrete time rules (implemented as grammar) incorporate cell birth, differentiation and death. The formalism also can be extended to account for geometry/morphology. This model is nice because it describes biological processes by means of stable states of dynamical systems instead of algorithms. However, the formalism is quite complicated and seems to be patched together in order to incorporate some of the phenomena seen in nature.

Dellaert and Beer[32] also appreciated the importance of incorporating development in the synthesis of autonomous systems. In their developmental process they model three levels: the multicellular level, the cellular level and the level of molecular biology. Unlike Mjolsness et al. who used a continuous time dynamical system, Dellaert and Beer model the genetic regulatory network on the molecular level by a Boolean network. At the cellular level they realized two important properties: The cell has to have a physical extent and it has to be able to divide. Differentiation and division is initiated by the molecular level. At the multicellular level cell to cell interaction and induction is mimicked by combining neighboring states by a logical OR to form new states. Using this model they evolved a simple agent exhibiting the relative placement of sensors, actuators and control systems. Unfortunately, the authors have not demonstrated the development of an artificial organism performing some kind of function.

It seems to be a very difficult task to unite theories of development and formal theories of self-reproduction. Each theory focuses on a different level of abstraction. However, since we want to investigate development of systems from a non-algorithmic viewpoint, we need access to the very low level phenomena of a cell model, but on the other hand, we don’t want to deal with artificial chemistries: A problem unsolved, but already recognized by Fontana and Buss[41].

Considerations based on the nature of cortical development have been applied to model and refine neural network architectures. Rolls and
Introduction

Stringer[97] demonstrate the convergence of a genetic algorithm used to tune a fixed topology network inspired by physiological findings to perform a certain computational task. In their experiment they aimed for the tasks of pattern association, association and competitive learning. The genotype-phenotype mapping used therein directly encodes cell and synapse properties including selective parameters such as the choice of the learning rule or ranged variables to describe the spatial connectedness of single neurons. The preparation of the network is performed by a genetic algorithm recombining and point mutating gene-sets probabilistically distributed based on their fitness. This approach agrees with the property of nature carrying far less information in the DNA of a species than in a full-grown brain by virtue of the fact that one single gene codes for properties represented in many neurons. By means of a genetic algorithm this approach mimics aspects of the evolution of the brain but does not attempt to explain how nature is able to develop a system of such a high structure from scratch.

Indirect encoding of networks have been also investigated in the theory of artificial neural networks. When using genetic algorithms, a difficulty arises in the choice of the genotype-phenotype mapping. A direct mapping has a problem with scalability: As the system’s size increases the description size requires non-linearly more space. This was realized by Kitano[64] who used indirect grammatical encoding of graphs for neural network design. In this scheme, instead of specifying directly the complete topology within the genes, a grammar for generating the network is encoded. Kitano uses quite simple rules based on matrix rewriting by substituting symbols. More sophisticated techniques for graph generation have been studied, for example by Gruau[52] who introduces rewriting directly on graphs instead of its connection matrices. He also considers modularity by allowing the insertion of already evolved subnetworks.

Nolfi and Parisi [85] model the development of neural networks by embedding them into space and letting their axons grow. They develop a set of genotypes such as branching angle and segment length which will influence the development of the organism. To test their model they demonstrate symmetry breaking and heterogeneity from homogeneous patterns. However, this phenomenon has already been captured in Turing’s study[114]. In our search of the under-
standing between local genetic mechanisms and global function, the presented setup seems too far reaching. Instead of extending the investigation to all biological phenomenas we would like to be able to just distill some of the crucial features.

In the field of ‘artificial life’, recent examples of artificial developing multicellular systems have emphasized the feedback interaction between a genetic description and environmentally derived signals, so elaborating the original von Neumann simplification of an entirely feed-forward Description. Various gene/environment models have been tested. Dellaert and Beer\[32\] used Random Boolean Networks to describe the genetic dynamics, whereas Fleischer and Barr\[39\] described the dynamics by differential equations. Eggenberger\[35\], and Bongard and Pfeifer\[19\] have used models of genes that incorporate regulatory regions and differential gene expression, similar to what we will present in this thesis. The book by Kumar and Bentley\[69\] presents a variety of further developmental models and reference \[108\] also provides an overview. The role of environmental feedback and homeostasis in gene regulation networks has also been described in a study by Quick et al.\[94\]. They demonstrated the evolution of a genetic control system that regulates environmental signals. The behavior of their system emerges through immediate regulatory reactions to the current state of the environment.

In general, these previous implementations have relied on genetic algorithms to discover suitable genetic instructions. Beside the inherent resource problems entailed by genetic search, the resulting instructions are usually phenomenological and so lack functional explanation. Also, in the most of the studies presented so far the organisms or networks were developed ‘offline’: They were first constructed independently of their environment by either a direct mapping from genes to properties of the cells and their connections or indirectly encoded and generated by a growing algorithm. After termination of their development they were confronted with their environment and thereby tested on their efficacy and performance, i.e. their fitness. However, the organism is embedded in the environment already throughout its development. Clearly, it can not completely ignore its influence. Moreover, constructivists believe\[93\] that the construction should be guided by the, throughout development already present sensory input and the current dynamics of the still growing network. In this case a fusion of both genetic grammar and dynamical properties of the intermediate circuit have to be addressed simultaneously and incorporated in one scheme.
Chapter 2

Complexity, Machines and Organisms

Natural as well as modern artificially created systems of this world exhibit a high degree of complexity. When we speak of complexity, we do not mean merely the number of involved components in a system, but rather the way in which they interact with one another. Given the complexity of the interrelations it should almost come as a surprise that these systems are as stable as they are. Although natural systems are quite different to artificial ones, they share a striking similarity: They are all organized in modules. In this chapter we will see that modularity bares many advantages, not just for the stability of the compound system, but also for its design.

2.1 Complexity

Complexity has equally fascinated and puzzled the scientific community of the last century. Even though humans seem instinctively able to qualify a complex system as such, no satisfying objective formal definition has so far been found. As stated above, the complexity of a system can not simply depend on the number of components it consists of: a crystal that has the same weight as a rat brain contains about the same amount of components, but is obviously less complex. This fact is reflected when considering the length of their descriptions instead of the number of components, because the descriptions also need to account for their relationships. A crystal simply requires a short description of the geometry of the atoms and the global shape while a nervous system of the same weight needs much more information to specify its complete structure.
2.1.1 Measures of Complexity

One of the most notable measures for complexity is the Kolmogorov complexity or algorithmic information content (AIC). This measure originates in Kolmogorov’s studies on algorithmic approaches to information theory [65] but was called to live in its current form by Chaitin[24, 46]. The strength of this measure is its objectiveness: Using the powerful concept of universal Turing machines and algorithms, for the first time there was a formal definition of complexity independent of the arbitrariness of language and human interpretation. The AIC of a sequence of bits is defined as the length of the shortest Turing machine program outputting the given sequence. This definition corresponds to the ‘intuitive’ definition of complexity when comparing the measure of random (coin-tossed) sequences to patterned sequences which can be produces by algorithmic loops or recursive calls: Random sequences, because unpatterned, seem more complex, and indeed, their AIC is close to the length of the sequence, because the only algorithm producing such a sequence is a program reading the stored sequence from its code and outputting it. Sequences with AIC close to their length are thus called algorithmically random. In most practical cases, however, it is non-trivial to compute the AIC of a given sequence of length because the minimal algorithm needs to be found. The search space is in the order of magnitude of \(2^n\) because we know, we can always find an algorithm with length \(\sim n\) by simply storing and reproducing the sequence in the program.

We say, a structure is \(c\) bit compressible, if its explicit encoding would require \(n\) bits, but its minimal encoding only requires \(n - c\) bits. There are \(2^n\) structures with explicit description length \(n\) bits, but only \(2^{n-c}\) descriptions of length \(n - c\) bits. Hence, at most a fraction of \(2^{n-c}/2^n = 2^{-c}\) of all the structures of explicit size \(n\) can be compressed by \(c\) bits. Therefore, not all structures can be encoded with less bits than their explicit description length. In fact, if some structures allow a compression, most other structures will require more bits than their explicit encoding length. If we want to exploit compressibility of structure and function, we will have to find a scheme in which the compressible structures are exactly the ‘useful’ ones. Through evolutionary pressure, Nature has optimized its description and construction mechanisms to be of this kind.

Grassberger[50] proposes complexity measures on strings of a finite alphabet. According to his theory, the minimal automaton that can recognize such a string gives an indication for its complexity, or rather for the complexity of the construction process of the sequence. Furthermore he proposes to use the information required to optimally predict a symbol of
2.1. Complexity

the string given the past sequence of symbols as a complexity measure, the true measure complexity.

Bennett[17] further refines the complexity measure in the context of Turing machines (as was the AIC) by including a computation time component and calls it logical depth. An object’s logical depth is related to the time that is required by a universal Turing machine to generate it from an input that is algorithmically random. What is nice about this measure is, that it attempts to capture the trade-off between the description code (i.e. the Turing program) and the time required to construct. As we saw in 1.2, there can be several ways to achieve a structure; by complete prespecification and straightforward construction or, on the other extreme, by almost no prespecification and a very long trial and error computation time.

While the presented measures are good theoretical frameworks for the quantification of complexity of pattern in abstract systems, they are unfortunately not very well suited for natural systems, systems whose abstraction is non-trivial and often observer dependent. Table 2.1 lists some of the reasons why.

To quantify behavioral complexity, Changizi[26, 27] made an attempt to estimate the size of the behavioral repertoire of species and managed to relate it to brain size and number of muscles. However, such a measure seems to depend very strongly on the experience of the ethologist that was categorizing the behaviors – it is far from objective.

2.1.2 Complexity depends on the Interpretation

While its rigorous definition is a big plus in favor of the algorithmic information content, it fails to capture all facets of what we perceive to be complex. Following Grassberger’s[50] reasoning, great randomness doesn’t necessarily mean great complexity. As illustrated in figure 2.1, the pattern in the middle is perceived to be the most complex one, while the AIC of the right one is maximal.

If we allow ourselves knowledge, on how these patterns are generated, and how they are interpreted, we can present simple algorithms for the two exponents, the regular one on the left and the completely random one on the right. But we can not easily find a simple algorithm producing the middle pattern. The left image is produced by an algorithm simply alternating the values of each pixel, when stepping through the pattern pixel by pixel. Since the interpretation of the pattern on the right is simply that it is random, the algorithm for producing such patterns is very simple too: for each pixel, with probability 0.5, choose the value 1, or 0 otherwise. This
Natural systems are not like static discrete patterns. They perform functions that are not measurable by a structural complexity measure. To measure function, usually Turing computational complexity is used. Continuous variables such as positions would require infinite precision, unless we only specified abstract relations between objects. This however yields the next problem:

There is no objective abstract description. Description and modularization of organism is ambiguous. Either every detail is specified down to the last atom, or prior knowledge about the function of modules is required.

Natural systems are part of their environment. Their function is product of structural relations. Object borders are structural properties. Objectness and Systemness are nontrivial to define objectively. Unlike Turing/von Neumann type architectures – there, the problem domain lives in another world than the computation. Traditional programs allow environmental interaction only through symbolizing interfaces and pre-specified sensors.

Natural systems can modify themselves. Because natural systems are part of their environment, they can modify themselves, or have the environment modify them. Classical complexity measures don't incorporate this trait. Turing machines, for example, are unable to change their transition rules once programmed.

Input/Output ill-defined and continuous. Natural systems extract input from their environment – stated differently: their organization is perturbed by interactions from the environment. They are constantly confronted with environmental influences. Most of their computation serves the homeostasis of their internal variables and organization when confronted with variations in the environment. Traditional Turing complexity fails to capture and measure this quality.

Behavior and survival are difficult to formalize into a Turing-type computational problem. They are not input-output relation problems, but rather involve actions invoking reactions from the environment in order to maintain a functional organization. Also, goal directed behavior.

No objective definition of the dichotomy between development of the organism and the final computation it performs. The system develops in the same world as it will perform the computation: it is the same mechanisms that drive the development and the computation. So how is the development different to the computation? Computation is, like development just a reaction to perturbation from the equilibrium?

Table 2.1: Some reasons why complexity of natural organisms is difficult to measure in terms of Algorithmic Information Complexity
2.1. Complexity

Figure 2.1: Patterns of different complexity (adopted from Grassberger[50]). The left image is a regular pattern. The middle image illustrates the behavior of convergence of the Newton-method for finding the roots of the function $x^3 + i$. The right image is a pseudo random pattern. All images contain approximately the same amount of black and white pixels.

process is very unlikely to reproduce the exact pattern of figure 2.1, but if all it needs to represent is random data, then most other outcomes will suit perfectly well. If, however, in the seemingly random pattern, each pixel represents a piece of data, interpreted by a complicated machine, even the switching of a single pixel might completely distort its carried information – this could be the case if the pattern represents a cipher key or a maximally compressed data file. Hence, the complexity of a structure depends critically on its interpretation. Thus, there can not possibly be an objective measure of complexity without considering the machine that produces it, or the machine, that interprets it. We can only quantify and compare the complexity of structures all interpreted by the same machine. It is the interpreting machine’s function that lends a structure its complexity.

The pattern in the middle of figure 2.1 is perceived complex by humans because we interpret a functional role in the neighboring relations. We can clearly observe that the value of each pixel depends on the values of its neighboring pixels, but we can not easily extract the rule the values follow. We perceive a structure as complex, if it is non-random, certainly not regular, but rather something in between. Because it constitutes a non-random structure that is highly non-uniform, we tend to accredit it a complex function if not design by an intelligent creator. The interpretation of neighborhood importance, which is fundamental to visual perception in our world, adds a functional component to the pattern – we can assume there must be a function due to its structure, but can not reason easily what it would be. Because physical laws obey the same neighborhood relations of local-
ity, this interpretation is in most cases valid. On the other hand, if we look at the machine code of an algorithm, the relations between neighboring bits will follow very different rules which will not seem as obvious to a human observer. Nevertheless, some programs can be a lot more complex than others notwithstanding their superficially similar structure in machine code. Such a distinction could possibly be better made by looking at the machine code in chunks of bytes or words rather than bits.

2.1.3 How does Complexity emerge?

Much of the fascination for complexity in natural things originates not just in the fact that it is there, but also in the question where it is coming from. In Nature we often observe simple processes cooperating to create a complex system, capable of achieving a higher level structure or function. Even though, in principle, we understand the underlying physical rules guiding the time evolution of each single component as a function of its state and its relations, we are unable to grasp how their interplay results in such complexity. This phenomenon is sometimes called emergence and denominates the often cited quote ‘the sum is more than its parts’. There have been many attempts to study this phenomenon\[36, 84, 54, 92, 63, 29, 12\] mostly in the dual discipline of self-organization which is concerned about the organization of structure from randomness.

From a broad view, in Nature, there seem to be two distinct but interplaying mechanisms responsible for the creation of complex structures: First, complexity hides implicitly in the structure and rules of the universe. Second, in development and evolution, the emergence of complexity can additionally be supported by the influences from the environment the system is embedded in. In this thesis we will see examples of both.

Complexity, implicit from the Structure of the Universe

The structure of the universe and its physical laws compose a clockwork driven by the perpetuation of time. The structure is locally homogeneous and isotropic, there is no distinguished location or direction in the universe. The physical rules also are homogeneous and isotropic, and even stronger, they are local. Aside from some paradoxical situations\[37\], there can be no instantaneous interaction of distant loci. To the amazement of scientists, however, these structurally uniform physical laws, allow for the emergence of complex, non-uniform structures on a larger scale. The complexity hides implicitly in the clockwork: no additional information is needed for its emergence.
2.1. Complexity

Figure 2.2: An example of a simple deterministic system that exhibits complex behavior. The cellular automaton of rule 30. Every line is the direct deterministic result of the previous line following a very simple set of local rules. The emerging structure at the bottom of the figure shows a non-random but also non-uniform structure.

How simple rules can create complex structure is probably best studied in the example of fractals or cellular automata\textsuperscript{119, 126}. Cellular automata are an abstraction of the physical world, reflecting both the homogeneity and the locality of the laws governing its dynamics, but discretized in time, space and state-space (they reflect the structure of the vacuum by allowing each cell to be potentially occupied by a physical part in a specific state, or empty). Von Neumann chose them to elaborate his theory of self-replication to avoid unnecessary complexity brought along by the complicated nature of kinematics and geometry. They vividly illustrate how, in an extremely reduced universe, complex structures can arise from simple rules (see figure 2.2).

A more biological example of a pattern emerging out of simple rules are reaction-diffusion systems\textsuperscript{114, 47}. Such systems show the spontaneous formation of patterns in a field of homogeneously distributed, identical chemical reactors. The local reactor dynamics are usually quite simple but give rise to nontrivial global patterns. In theoretical biology such systems have been used to explain animal coat patterning\textsuperscript{83} and socio-economical patterns across populations of species.

**Complexity from Environmental Influences**

If the organization of a functional system in contact with an environment has to be stable throughout a large number of fluctuations from its environment, Ashby’s law of requisite variety\textsuperscript{10} requires that a system needs at least as many internal variables as there are independent influences from
the environment. From this law it follows automatically that a homeostatic system that is in structural interaction with a complex environment needs to compensate for this complexity by its internal complexity. The internal organization hence reflects the degree of interaction with a complex environment. We claim that parts of this complexity can be acquired through extraction from the environment during development. In Chapter refnetwork we demonstrate how an entire model can construct itself from observation of the world.

This acquisition of complexity can also be observed in Nature: In early stages, developing systems are very fragile. They can not handle largely fluctuating influences. Thus, eggs and the mother's womb and later the social environment of the family shield fluctuations from the outside world and provide an environment with reduced variety. Most natural organisms blindly bootstrap in non-complex, quasi-constant environments to a degree of complexity high enough for them to face the real world outside their breeding space. A large portion of their complexity must therefore be provided a priori by their genetic code and the developmental process as discussed in the previous section. Then, later in development, when the system has developed sensors (visual, tactile, etc.), will it acquire information from the environment.

The inability to compensate for small environmental influences could also be regarded as the driving force for the development itself. Maturana and Varela[78] describe this as dynamic equilibrium. In their point of view, there is no development, only reactions to environmental influences that can be interpreted as perturbations to the dynamical system of the developing organism (Similar to the interpretation of Figure 1.2). It is these perturbations that drive the development further.

**Evolutionary Complexification**

Heylighen[56] studied the growth of complexity from an evolutionary point of view. He argues that more structural complexity gives rise to more niches and hence we have an accelerating structural complexification. He makes the analogy to adding puzzle-pieces together and so increasing the border, thus adding sites to add pieces. Now, we need more functional complexity if we have to cope with the greater variety of environmental variety. Or vice versa: A species capable of surviving a larger variety of circumstances is fitter (if it can for example eat more diverse food or live in other climate conditions). However, to achieve this, it needs a greater functional complexity.

There is a trade-off: The larger the variety of available options, the
more difficult it will become for the control system of the organism to select the most adequate one. Hence, the evolving systems will asymptotically reach a trade-off level, depending on the variety of perturbations in their environments, where requisite variety is in balance with difficulty of decision-making. For example: parasites need only be as complex as to be able to deal with the (already partially homeostatic) environment of their hosts. Self-maintaining systems exist at different levels of complexity. They all must satisfy a closure relation: They must be structurally stable and fit in the sense, that they can cope with their functional variety in a way justifying the cost of their higher structural complexity and the complications of decision making.

The complexification can also be explained in the light of co-evolution of systems and their environments. An increase in the variety of the environment to one system is compensated by an increase of functional complexity. This feeds back as environment to the another system and so forth. This fact has also been described in the “Red Queen Principle”[115], which states that a system must continuously develop in order to merely maintain its fitness relative to the systems it co-evolves with. Saunders[101] notes that it is generally easier to add a component to a system and maintain its operation rather than taking away a component and so disturbing the complex network of interdependencies. He then reasons that it seems natural that evolution finds more complex structures rather than simpler ones as time goes by.

2.2 Machines and Organisms

The ability of mankind to construct tools has provided us with a huge advantage in the process of natural selection. Since the first emergence of this quality, the techniques have been largely refined and brought to perfection: Our tools have evolved to computing machines and robots. Their designs consist now of modular components encapsulating functionality assembled together to fulfill a higher level function. All the design and construction processes are organized in a way, that a single human being only needs to consider a limited amount of information at any particular moment: Modules have simple interfaces for the functions they provide, protocols allow for abstraction of difficult communication processes, etc. In western thinking, dealing with sequences of simple causal relations is extremely pronounced and formalized in the notion of algorithms. Algorithmic thinking allows to solve a problem in a sequential fashion while always only considering a very limited range of variables. It is no surprise
that the current computing machines process their information in exactly this fashion. They are very well suited to our thinking. Or maybe, we are very well suited to handle current computers. Figure 2.3 illustrates this principle.

2.2.1 Engineered Modules

Our design and construction of physical systems is not as sequential as the design of algorithms for computation, but it follows a hierarchical structure: Modules are assembled to super-modules, and those in turn again to higher level super-modules. At each level of the hierarchy only the interfaces of the modules on that particular level must be considered, their internal function is entirely encapsulated by the module. This way, the complexity of interactions to be considered for a particular function is limited to the complexity of the interactions among the involved modules (figure 2.3).

Integrated circuits are built from logical gates. Logical gates are assembled to higher level logic, like flip-flops, binary adders, multiplexers, memory registers, and finally CPUs. CPUs have well defined communication protocols and use those to communicate to external memory, to the IO subsystem and so forth. The same applies to engines that are created from mechanical parts and bolts, or large software projects that use objects and design patterns and libraries. All these modules are designed to encapsulate their internal function and shield it from the outside to only export
functionality that needs to be accessed from higher level processes.

The trick that finally allows the use of such modules easily are stand-
dardized protocols and interfaces. They allow to plug modules together
without having to worry about their internal workings. It was the stan-
dardized railroad gage that allowed transportation of products across an
entire continent in an efficient way, that helped the industrial revolution to
engage fully. Later, it was due to standardization efforts, such as stan-
dardized screw threads\cite{123,103} or the common voltage levels across
early digital components that allowed engineers to easily construct systems
built from components provided by many different manufacturers and so
allowed technology to grow as fast as it did and still does.

2.2.2 Natural Modules

Similarly to artificially engineered systems, natural systems are also built
from modules. Modularity can be observed on many different scales of
organism: Cells constitute tissues that constitute organs that constitute
systems. All of which have clearly defined functions, protocols and
interfaces\cite{30}. As opposed to engineered systems the interfaces are more
complicated and we do not yet understand the protocols fully – and the
wiring of the component over-strains the power of human comprehen-
sion.

Not just are natural systems modular, they also constitute a hie-
archy: Simon\cite{105,106} describes the hierarchical quality of biological complexity
in his article “The Architecture of Complexity”. He claims that their func-
tion can be collapsed into a nearly decomposable network whose interaction
matrix resembles a block-diagonal matrix in which all non-block elements
are smaller than some $\epsilon$. Thus the global dynamics can be explained to a
large extent by the dynamics of the subsystems and some small interaction
terms. More recently, modularity and hierarchy has been found in many dif-
ferent aspects in natural organisms\cite{55}, such as in gene circuits\cite{117,116}
as well as in metabolic networks\cite{95}.

In Nature, there are several reasons why modularity makes sense: First,
modular systems are much less fragile and sensitive to mutations because
a single mutation only effects its module and not the entire system. This
mechanism allows for the independent evolution of (orthogonal) modules
because the configuration space of the whole system is decomposable into
almost independent components. The search strategy for finding the opti-
mal configuration is thus reduced to the search of the optimum for each
component independently: The global optimum is achieved when each
module optimizes the fitness of the whole system relative to the current
configuration. This means, nature can \textit{turn the knobs separately} to fine tune each module and still reach a near optimal solution.

Second, limiting the interaction between components and thus the complexity of the system will enhance the stability of the system: By the law of requisite variety\cite{10}, a system with many free input variables needs to incorporate much higher internal structure to be able to maintain homeostasis. It is thus a better strategy to encapsulate functionality in modules and only export and import information nonspecific to the modules internal function, but rather to its control. It is much easier for the modules to internally maintain their stability and correct for perturbations from their environment than it would be for a complexly interleaved system\cite{107}.

A further reason for modularity in natural processes is that reusable modules save description code. Once encoded, they can be used many times with a minimal cost of description.

2.2.3 Differences in Organization

From a system's level perspective and in terms of modularity, natural organisms have similar traits as machines\cite{30}. In many other ways, however, they are still very different. The most prominent difference is probably the fact that natural organisms are essentially self-constructing and self-maintaining. Being able to self-construct allows Nature to employ profoundly different mechanisms of design and construction than human engineering. In particular, by contrast to engineered systems, feedback loops in natural systems are extended to include structural organization: Instead of just regulating signals, natural systems can \textit{regulate their structure} through a feedback of local sensing and local construction. Through this capability development and self-maintenance is greatly enhanced. Following Matutana and Varela's reasoning\cite{78}: Natural systems are homeostatic with respect to their organization – or differently stated, the final organization of an organism is a stable equilibrium maintained by the active process of local construction.

Even though in principle modularized, Nature increases the complexity of its species by first building the modules but then letting processes interact across the boundaries of the modules at different levels of the hierarchy: In the beginning of development, for example, a rough body plan is laid out. Later, the placed components start to interact bilaterally\cite{127} and are so increasing the complexity at the different levels of the structural hierarchy.

Nature employs very specialized types of cells; nevertheless, the basic
2.3 The Limits of Engineered Design and Construction

architecture is common for all cells. The functionality of an entire organism is not produced simply by the compound function of its specialized components, but much more by the way they are organized with respect to one another. This is in contrast to engineered construction, where complicated and very specialized machinery is encapsulated inside modules and not allowed to interact outside their thin communication protocols.

2.3 The Limits of Engineered Design and Construction

In his book 'Notes on the Synthesis of Form'[8], the architect Christopher Alexander distinguishes between two different types of processes for the construction of form: unselfconscious and self-conscious processes. The unselfconscious process is employed by primitive cultures, not aware of design principles and engineering techniques, but simply copying what has been done before and reacting to changes in environment by the most obvious, direct action. Even though individuals of such a culture might not understand the function and purpose of the structures they build, they know how to construct and repair them – thereby possibly optimizing its design by accident and leaving this information for future generations. The feedback that is refining the forms built is given by its structure itself with respect to the function it is supposed to fulfill. Unselfconscious cultures are only slowly changing but keeping in stable and robust equilibrium with their environment. By contrast, selfconscious cultures employ means of engineering, reasoning and communication to design forms to fulfill a specific function. Alexander claims that selfconscious systems are unable to create forms that fit well:

In the unselfconscious culture a clear pattern has emerged. Being self-adjusting, its actions allows the production of well fitting forms to persist in active equilibrium with the system.

The way forms are made in the selfconscious culture is very different. I shall try to show how, just as it is a property of the unselfconscious system's organization that it produces well-fitting forms, so it is a property of the emergent selfconscious system that its forms fit badly.
2.3.1 Too complex for human comprehension

Traditionally, engineering relies heavily on hierarchical design and construction. Hierarchical structures, however, suffer from a bottleneck\(^1\): As processes are becoming more complex, information from one side of the hierarchy needs to flow to the opposite side at significant rates (see Figure 2.3). Thus, a node sitting at the top of such a hierarchy will inevitably reach the limit of its processing capacity as hierarchies are becoming larger and problems are becoming increasingly complex. A way to circumvent such a problem would be to connect the two communicating processes through a direct communication line. This, on the other hand, will greatly increase the complexity of the structure. Engineers are thus now facing a trade-off between an overviewability of the structure and the complexity of problems such a system is capable of solving.

Modern engineered systems such as ones of the scale of the Airbus 370 mentioned in the Introduction or large software systems face this exact problem of limited bandwidth and thus need to add relations that are violating the overall hierarchical structure. But unfortunately, these relations greatly obstruct the maintainability of such systems.

Alexander claims that humans are unable to foresee all possible situations a system might encounter, and even if we were, such a system would be too complex for human comprehension to understand it let alone design it. Rather, he states, we should attempt to design systems in a way that they can be easily modified by their users and so converge to optimally fulfill their purpose and adapt in unforeseeable situations.

Even though Alexander was studying form and function in the context of architecture, the same principles apply to general engineering of any technology. In contrast to selfconscious processes where design and construction is clearly separated, in unselfconscious processes, the constructing element is an integral component of the system. In the case of architecture the design of a particular artifact was the emergent result of a long history of refinements in an unselfconscious culture. Here, we propose to incorporate these principles of incremental design in any engineering discipline.

Modern paradigms of software development aim back in the direction of unselfconscious design. Most prominently, Extreme Programming (XP)\(^1\) carries many hallmarks of Alexander’s unselfconscious processes: The design is incremental, the development tools are very advanced and optimized for ease of refactoring and entire restructuring of the code. Software units carry unit tests that know how to locally verify and guarantee the correctness of their function. This way the programmer does not have to keep everything in mind (which they simply could not do) but can safely rely on
the tests.

Here, we suggest to remove the constructor/programmer altogether and substitute him with a designer that specifies the final structure a priori. The construction (programming) is then carried out by the system itself. Throughout the construction process, the global structure and its function feed back onto the components and determines their further development.

2.3.2 High level languages and Compilers can help

To help shielding the complexity of an entire detailed description of a system from the human engineer, compilers and computer aided design systems can help a lot. Compilers take as input an abstract specification in high level language and compile them into low level instructions or descriptions. While traditionally compilers are used to translate high level computer programming languages into machine code, compilers can in principle also be used to generate low level descriptions of physical structures (for example VHDL) or processes of any kind.

In the most simple case such a compilation can be just a sequential substitution of the keywords in the high level description into snippets of low level instructions. This can happen on several levels of a hierarchy (for example, the Java compiler produces byte code which is then interpreted by the Java Virtual Machine by translation the byte code into machine dependent CPU instructions). In practice, the principle is the same, but the compiler first passes the description through a parser and uses a grammar to construct the low level code. In the discussed simple case, due to the reversible mapping between the high-level language, the grammar and the low-level language, the compiled instruction codes can be decompiled back into their original form in the high level language.

More sophisticated compilers however optimize their output code by careful bookkeeping of registers, reordering instructions, searching for redundancies in sequential code regions and collapsing them into a more compact form and pass the code through many more elaborate irreversible optimization steps. Code that is optimized in such a way loses its explicit decomposability into a hierarchical structure derived from its original description. However, because it derives from a hierarchical structure, it is still nearly decomposable (as described in Section 2.2.2).

Drexler claims, however, that if resources are scarce the industry is usually better off using human brainpower to design their systems by hand entirely instead of compilers that are unable to optimize as precise as human intelligence. On the other hand, modern compilers include opti-
mization techniques[7], likely to be unfeasible to humans – so it is arguable whether humans could ever live without automated compilation.

Compilers have proved to be incredibly useful so far. However, they output a static set of data which represents their best shot at an implementation of the specification given in the high level description. So far, due to its feed-forward assembling technique, compiler technology does not yet include any means of creating low level code that is actively utilizes self-modification. In fact, traditional compiler technology aims at outputting a complete explicit functional code by design. In a first step, novel technologies could for example ascribe some of the optimization stages to be part of the compiled code rather than to the compiler code.

2.3.3 Self-Construction requires novel Design principles

Ideally, we would like to be able to find an explicit mapping between the functional definition of our to-be-constructed system and the implicit rules of construction that would permit such a system to self-construct from a single seed. Such a mapping could well involve compilers as described in the last section. However, while a compiler can help in assembling the right instructions to a useful low-level description, it can not help in finding the right high level description.

Since the construction mechanism is entirely different to traditional passive exterior assembly, the traditional design tools will also reach their limits. The fact that Nature and its evolutionary pressures took billions of years to come up with the current species is not much comfort. On the other hand, evolution is the blind watchmaker[31] and not the engineer – so we might be able to find solutions a bit quicker. Evolutionary algorithms [40, 96, 57, 67] could help, but would have to search a vast space of possible genotypes by a biased random walk - still blind. We should be able to do better than that.

To tackle this problem, we propose to discover stable developmental modules. These developmental modules give rise to stable and self-maintaining structures that can be used as building blocks for the incremental self-construction (see Chapter 4). The final structure is then built as a cascade of sequential sub-objectives that are switched on conditionally on the state of its supporting structure. After the global development has completed, local processes can help to refine the structure, much like the optimization stage in the process of compilation seen in the previous subsection.

Unfortunately it is a nontrivial task to extract local rules to find stable
developmental modules. Also, it is not clear what the local processes would have to look like to refine and complexify the structure. A lot of effort will have to go into these questions. The design of the sub-objectives will depend on the modules and local processes and also require novel design tools.

2.4 Conclusion

Complexity is difficult to quantify. Nevertheless, no one would doubt its existence. Some people might put it off as: “humans are just too stupid to understand it”, but by saying that they implicitly agree that it is too complex for us to understand, but making the strong point that complexity depends on the observer, and the purpose. Although the complexity of artificially created systems is very impressive, it is not nearly reaching the dimension of naturally developed complexity seen in species. This is probably due to the fact that humans are getting close now to reaching their hard upper limit of complexity they can handle in engineering. By the capacity of our mind, we are limited in the dimensionality of the problems we can solve. Sometimes, we can disentangle the problems and distribute them onto different scales and levels in a hierarchy. While we can increase the height of the hierarchical pyramid and use computers to support us, we can not anticipate anymore the global dynamics of the entire system. Additionally, we saw that hierarchical systems have a problem with bandwidth in the higher level nodes as soon as problems increase in complexity.

In natural processes, complexity seems to grow and adapt from the complexity of the environment – by Ashby’s law of requisite variety, natural systems must reach a critical internal complexity to be able to survive. Wouldn’t it be great if we could use some of the natural principles for our construction? The first step to employ such principles would be for us to use self-construction in our systems. Without self-construction we can not possibly get access to the essential natural mechanisms, such as complexification and self-maintenance.

However, self-construction brings along a new set of problems. Apart from the fact that we do not yet know how to physically implement a self-constrcuting system, the design of a higher level system relying on self-construction would be a difficult task by itself: Since self-construction happens from the inside out, in contrast to traditional fabrication which had global access from the outside, we have to find local rules that give rise to the correct development. Furthermore, we have to find methods of design for the construction of larger systems that incorporate many different com-
ponents – how would such subsystems know of one another let alone communicate to one another? We propose a quasi-hierarchical developmental scheme, by which structures are built on top of previously built structures following a sequence of sub-objectives. We will see such an example in Chapter 4.
Chapter 3

Self-Construction

In the challenge of building self-constructing systems, there has been traditionally one school of thought: The kinematic feed-forward type of self-construction, brought to thought by von Neumann[119] in the 1940s in his description of a self-replicating machine. In 1959, Feynman[38] proposed a hierarchical machine shop in which factories build smaller factories, which in turn build smaller factories, and so on, until the lower limit in nano scale is reached. Von Neumann’s theory together with Feynman’s optimism about the creation of machines in nano-scale caused an enthusiasm about the possibility of building self-constructing machines. NASA[42] worked on a concrete example of building a self-constructing lunar factory and in 1986 and Drexler[33] even envisioned Grey goo, an end of world scenario in which molecular self-replicators consume all living matter of the world. All these proposed systems have in common that they are feed-forward in their description and construction – very similar to current design and engineering, they are translating an explicit description into a physical instance, with the simple twist, they they are self-constructing.

Only recently have researchers begun to think more organically in design and construction: Like in biological systems, our engineered systems should not translate an explicit description, but rather they should grow along local rules of construction specified in an implicit, incremental description. Very recent exponents of such self-constructing systems arise out of the field of synthetic biology[14] in which the self-construction of a synthetic non-trivial multicellular system is demonstrated.

This chapter discusses von Neumann’s original theory of self-replicating systems and then goes on to look at self-construction more generally and from the point of view of cybernetics.
3.1 Von Neumann’s theory of self-replicating systems

Shortly before his death in 1957 John von Neumann gave a set of lectures about work he had carried out in the 1940s, concerning the nature of self-replicating systems. His then formulated theory should later become the basis for most thoughts on self-replicating and self-constructing systems. Although he mentions a kinematic example, he seemed less concerned about the actual physical implementation of self-replicating systems but rather tried to distill the crucial features of abstract systems capable of replicating themselves. As Langton [72] realizes:

Von Neumann’s approach to the problem of self-replication was a classically logico-mathematical one: If self-replication is being carried out by a (highly complex) biochemical machine, then that machine’s behavior is describable as a logical sequence of steps, i.e. as an algorithm.

Von Neumann actually succeeded in building a cellular automaton with 29 state cells capable of self-replication and thereby found a logical organization sufficient for an automaton to be able to replicate itself. Hence, his work contains the crucial ingredients for a technology actually achieving self-construction. Figure 3.1a shows his original sketch of a self-replicating machine implemented in his cellular automaton. Figure 3.1b shows a snapshot of a more recent implementation of von Neumann’s self-replicator carried out by Pesavento in 1995. It actually works.

The universal constructor and the code-copier

Along the definition of Turing’s Universal Turing Machine, von Neumann introduced a universal constructor $A$, capable of producing a machine $X$ when fed with its description $I_X$. Such a machine would be able to duplicate itself by reading $I_A$ and thereby constructing another machine $A$. The machines, as well as descriptions of machines had to reside in the same physical substrate, i.e. cellular automaton in order to respect the true nature of naturally self-replicating systems. It was clear to him, that such a machine $A$ would not, by itself be able to replicate itself anymore because it was lacking its own description. Therefore he created a code-copier $B$, able to produce two copies of a description-code fed with.
3.1. Von Neumann’s theory of self-replicating systems

Figure 3.1: a Von Neumann's self-replicator with the description code ($L$) and the Universal Constructor ($CU$). The constructor is busy building a new copy of itself according to its description with its construction arm (original figure from [119]). b Pesavento’s implementation[91] of the self-replicator in a cellular automaton as proposed by von Neumann. The figure shows just the universal constructor. The fine line at the bottom is the beginning of the description code. The entire description is 84’000 cells long. The construction arm is at the top right of the constructor.
Copy by self-inspection?

Why shouldn’t there exist a machine $B$ capable of replicating not just a description $I_A$ of a machine, but the machine $A$ itself? The problem von Neumann saw was twofold: On one hand there is the measurement problem, stating that a physical system, when inspected needs to interact with the observer, thereby changing its state, rendering a different system. The states he uses as description states are exactly the states stable under measurement transformations, so called quasi-quiescent states. Quiescent states are, by definition, not useful to perform computation or construction because they are static – they don’t interact with neighboring states because otherwise the description would change. But a self-inspecting machine would have to have a means to look at functional (non-quiescent) states of itself and copy those. It was not clear to von Neumann, how such a process could be performed.

On the other hand he feared a logical problem along the lines of Turing’s halting-problem when allowing a machine to inspect itself. He never gave a rigorous proof for the logical case. In fact, in 1977, Laing demonstrated self-replication by self-inspection, thus nullifying von Neumann’s assessments.

The processor

Finally, von Neumann introduced a processor $C$. $C$ takes a description $I_X$ and feeds it to $B$. $C$ then feeds one of the two copies of $I_X$ to $A$ in order to construct the machine $X$ and the other copy is appended to $X$, yielding a machine $X$ together with its description $I_X$. The machine $A+B+C$, together with the description $I_{A+B+C}$ is therefore able to produce a new copy of $A+B+C$ with its description, hence self-replicating. While this description is abstract in terms of machines $A$, $B$, $C$ and descriptions, von Neumann provided rigorous implementation details to build a self-replicator in his proposed 29-state cellular automaton.

Some considerations relating to natural systems

Von Neumann’s system has been carefully designed to incorporate the function of self-replication. As it stands its components can be clearly classified into a functional machine, the description of a machine and the world the whole system lives in. Additionally, by construction, the self-replicating system itself can be subdivided into the constructor, the code-copier and the processor. It is conceivable, however, that the substrate this world con-
stitutes of and the physics ruling the time evolution would allow a more compact but less modularizable system to self-replicate as well. Inspired by the way nature acts we would still expect this system to contain some kind of a description and some other kind of functional system, much like the genes the molecular machinery that processes them. It is quite remarkable that von Neumann came up with his description of self-replication including a code-copy mechanism and a universal constructor that mimics RNA-polymerase, well before the discovery of DNA by Watson and Crick in 1953[122].

In a less modularized point of view, the entire world and its governing physics could take the role of a huge assembling machine and the descriptive code would merely influence the way the world assembles, thereby creating a system of the described kind. From this perspective, the code to would be functional: The code or a part of the code (via transcription) gives rise to some catalytic action on the environment thereby possibly increasing the complexity of the surrounding structure. The changed surrounding structure conversely induces new transcriptions of the description adding new structure and so forth. When the system has completed its construction, two possible scenarios are imaginable: The surrounding structure stops inducing transcriptions and hence the system stops developing or, and nature seems to act this way, the system converges to a stable feedback loop between the code-transcription and its surrounding structure, not changing the overall structure and function of the system anymore. This stable but unsteady system would implicitly be able to react immediately to changing environmental influences. The transcription feedback loop then takes the role of a live-supporting system.

3.2 Construction with Feedback

Traditional explicit construction is a feed-forward process. In this section we investigate the possibilities we gain, with the technology of self-replication. Because a self-replicating system can be sensible to the environment, it can introduce a feedback loop with the structure it is constructing. Understanding of such principles will certainly enhance future construction. Similar to the way complexity arises in natural systems as we saw in Section 2.1.3, construction can employ these principles to build systems.
3.2.1 Traditional feed-forward construction

The traditional explicit construction is sketched in Figure 3.2. A universal constructor reads an explicit description \( E_F \) through \( E_N \) and instantiates the described functional automaton. The constructor sequentially maps each description state to its uniquely defined functional counterpart. This process is open loop in the sense, that the constructor doesn’t need (and is not able to) check what is there already. It merely maps description states to functional states. Once the automaton is completely constructed, it commences its function. This type of construction is also used in von Neumann’s self-replicator as described in the last section.

The figure illustrates also the fact that the description code lives in the same world as the constructor. However, by itself, it has no function except the storage of the description it carries in its structure. Von Neumann termed these type of states quasi-quiescent, i.e. their function is to do nothing but remain in the same state. Quasi-Quiescent states are comparable to stable physical structures like for example the nucleotides on DNA or the ferromagnets on harddrives.
3.2.2 Implicit description without environmental input

Figure 3.3 illustrates an example of an implicit description encoding a function. In this case, the description code merely specifies factories which through interaction with themselves and the already built structure construct the final function.

The constructed factories could in principle also create helper-structures used for the final construction. For example, it might be more sensible to construct a sine number generator for later use instead of encoding an entire sine-table in the description. A construction of this kind could in an extreme case also trial and error the physics of the world by building all possible automata (or random ones) until it finally finds the correct instantiation of the function. Evolutionary algorithms are of this kind. Of course this takes a long time to converge. As discussed in chapters 1 and 2, it is a trade-off between description size and development time in the hands of the engineer to decide.

For complex systems, we expect this encoding to be shorter than the explicit encoding: In complex structures, where lots of similar but not identical modules are needed, it will pay off better to first build a factory that knows how to compute and construct specific modules when needed in the construction process, rather than a priori specifying all occurrences and all details of all modules. A state machine can for example be used to map the current state of constructed system to the next action to be taken. Such a description would encode only milestones and how to get there, or what decisions to take at ambiguous forks of the path of construction. It would therefore resemble algorithms or flowcharts rather than an explicit blueprint. For example: When building a wall, instead of specifying the location of all bricks, it is more sensible to describe the mechanism of tiling the bricks and telling the factory to stack the bricks until reaching a certain height. In Chapter 4 we will see a non-trivial example of such a system.

The way such a system builds structure from a starting seed is similar to the mechanisms Nature reaches complexity as discussed in Section 2.1.3 Complexity, Implicit from the Structure of the Universe.

3.2.3 Implicit description with environmental input

The implicit description scheme presented above narrows the space of phenotypes to the class of self-constructable and functional types. It thus incorporates a deterministic mapping between genotype and phenotype. A given genotype will always develop into the same phenotype – in that sense, self-construction is just a quite sophisticated mapping between genotype and
Figure 3.3: **Implicit description.** In this case, instead of constructing the final functional structure, the constructor \( C \) constructs further factories \( F_1 \) through \( F_4 \) corresponding to their description \( I_1 \) through \( I_4 \) (1 and 2). 3. Through local interaction the factories jointly create the final functional structure. This construction requires a shorter description and a less complicated constructor. 4. The final structure at work.

phenotype space.

But self-construction allows a further refinement of the process. Since the developing system is at all times functional and in constant contact with its environment, it should and probably can not neglect the influence from outside. Generally speaking, a developing system can query the environment and incorporate its structure. Its final configuration will thus depend on the environment it was born into. When allowing the environment to influence the developmental process, systems do not need to contain a recipe for all possible states they could encounter and their counter-measures anymore. The development process itself will take care of it by attempting to retrieve this information from the environment.

Figure 3.4 illustrates this process construction. It is very similar to the construction from implicit description of the previous section, but with the important difference that in this process the assembling factories \( F_1 \) through \( F_4 \) are already interacting with the environment and can so react specific traits of it by building the most suitable function \( FUNCTION' \) for this particular environment. Event though the process is illustrated from left to right in the figure, in self-constructing systems, the code \( I \), as well as the constructor \( C \), the built factories \( F \) and the resulting function
3.2. Construction with Feedback

Figure 3.4: **Implicit description, with environmental interaction** This case is similar to that of Figure 3.3. However, in step 3 the constructed intermediate factories not only interact with themselves to create knowledge, but they are also allowed to retrieve information from the environment they are embedded in. Using this strategy, without explicitly encoding all possible scenarios the system could encounter *a priori*, a system of this type can adapt its function to the environment while it is developing, essentially choosing its appropriate function from a larger family of functions. In the figure, the system develops function $FUNCTION'$ to react to inputs of the kind Input'.

$FUNCTION'$ are all part of the same system and will not be separated.

In Chapter 5 we will refine the self-constructing organism of Chapter 4 to incorporate this process. Chapter 6 shows another attempt to build such a system.

### 3.2.4 Four feedback loops that control development

Because there is no global observer and outside assembler in self-construction, such systems face the problem that they do not see what it is they are building and they can not randomly access any location of the system and place a component there. Rather, such systems need to be able to locally extract information about the global structure and use it to guide its local construction.

The control of development can be categorized into four feedback loops (Figure 3.5): A local structural (L1), a local functional (L2), a global struc-
Figure 3.5: The feedback loops that control the development of a system. All developmental actions (construction) are local structural in their nature, but they can be controlled by more than just local structure.

- **L1 Local structural (Physical constraints)**: This feedback loop represents local structural constraints like, for example, physical properties, mechanical obstacles, but also guideposts that can lead a direction. Passively self-assembling systems\[128\] are of this kind.

- **L2 Local functional (Local learning)**: Local structure gives rise to functionality that can be used to steer further construction. Long-term potentiation in synapses is of this kind. Driven by the activity of the cell, structural changes in the synapse result in an increase of the synaptic strength.

- **L3 Global structural (Cooperative effects)**: Local structure and function can give rise to global phenomena that can again feed back locally, as discussed in section 2.1.3. These signals can be very useful to build nontrivial higher level structure while using only local construction mechanisms.

- **L4 Global functional (Reinforcement learning)**: Finally, the global structure gives rise to a global function, such as a behavior of the system. The reaction of the environment to such a behavior can also be used to control development. Such an environmental reaction, however, is very high-level. It could be, for example, a reward signal that depends on the behavior of the system over an extended amount of time.

These loops can aid development, but not in an obvious way: The two local loops are blind to global structure and function, so how should a local
construction mechanism know what actions to take to ultimately reach the correct global structure. The local feedback loops should therefore only be used to implement some mechanisms in local processes such that the global feedback loops do not need to worry about them anymore. They can for example maintain a homogeneous structure and function, and locally increase complexity. The global loops are responsible for the larger layout of the system. The difficulty in the design of the global loops is understanding how global emerging signals can be used to command local construction.

Ultimately, construction is a function that performs a change in structure. This change of structure will further result in a change of function, thus refining the way the system continues constructing. This feedback loop is now closing the circle between structure and function. While it is easy to deduct function from structure (just look at it, or simulate it), the other direction is usually not trivial. To find the machine $M$ that performs function $X$ has been reserved for human creativity until now. But the global functional control loop does exactly this. Depending on the global signal that reflects the function of the system it tries to find a structural change that will optimize the function. These loops are closed, hence a system of this kind can in principle reach an equilibrium state. An equilibrium state that will be both structural and functional – the entire organization is thus self-maintaining.

The control loops can further on fulfill other important jobs: They can react to environmental changes by slowly shifting their limit cycles or attractors to appropriate neighboring trajectories that are stable under the influence of the new environment. This way the stability of the system is not algorithmically enforced but implicit in the very nature the environment acts upon the system and therefore on its control loops. It is also conceivable that in certain environments more than one stable equilibrium exists. Transient changes or tonics in the environment can then give rise to a switch between the stable states. Different attractors will also result in possibly different morphologies and different functions of the system – the biological phenomenon of cell differentiation works just like this. With the control loops the system can act and react. Even reproduction can be initiated by the loops: The system is alive.

### 3.3 A cybernetic view on self-construction

We can look at construction and self-construction in the context of cybernetics. Cybernetics is the theory of control and regulation and allows an abstract view on modular systems [124, 10]. Figure 3.6a shows two simple
circuits of typical cybernetic systems. The left diagram shows a simple ballistic system that is subject to disturbances $D$ of the world. $T$ is the world our system is embedded in. $E$ is the set of essential variables of the system. For our system to survive, its essential variables need to remain in the set of acceptable states $\eta$. If $T$ is a non buffering system, i.e. its output variety is of the same magnitude as the input provided from $D$, but the variety of acceptable states in $E$ is smaller than the variety of output states of $D$, $T$ has no way of keeping the output in the acceptable range while facing the whole variety of $D$. The right diagram shows a regulatory circuit. The regulated system contains a regulator $R$ that is also receiving input from the disturbances of the world. The regulator can react to the disturbances of the world and influence the action of the world $T$. Essentially, $R$ is blocking the variety of $D$ through $T$. Ashby’s law of requisite variety\cite{10} states, that $R$ can maximally reduce $T$’s output variety by the amount of its own output variety.

The favorite example of cyberneticians for a regulatory system is a thermostat. In this example, $D$ is the outside temperature of a room. $T$ is the physics governing the temperature of the inside of the room and $E$ is the temperature inside the room. In the ballistic case, $T$ acts as low pass filter and will adapt the inside temperature to the outside temperature provided by $D$ until the thermal equilibrium is reached. If the room needs to be in habitable conditions, the set of acceptable states $\eta$ might be defined as the temperature range between 18°C and 23°C. Because the system has no means of correcting the temperature its essential variables are under full

Figure 3.6: **Left:** A diagram of a simple ballistic system. $D$ is an exterior disturbance applied to the system. $T$ are the rules of the world governing our system. $E$ is the set of *essential variable* of the system. $E$ has a set of acceptable states $\eta$ and a set of inacceptable state $\hat{\eta}$. Our system dies, if its essential variables are in an inacceptable state. **Right:** A regulatory system. Additionally to the forward part of the left system, this system contains a regulator $R$. The regulator also receives input from the disturbances $D$ and can react accordingly to them and influence the action of the world $T$. The aim of $R$ will be to keep the essential variables inside $\eta$. 

control of the disturbances. A regulator $R$ could act as thermostat by heating if the disturbance $D$ is lower than 18°C and cooling if the disturbance is higher than 23°C. The regulator can thus make sure that the essential variables remain in the set of acceptable states $\eta$.

Any machine acting in the world can be viewed as a regulator $R$. For example, if we want to look at a machine that is computing a function $f(d)$ of some input $d$ provided by the world through its disturbances $D$, we could define $T$ to be the verifier of the function by performing the action $f(d) - r$, where $r$ is the output of the regulator. The set of acceptable states $\eta$ would then be defined as an interval $(-\epsilon, \epsilon)$ for small $\epsilon$. The only regulator that could then keep this system within the range of acceptable states $\eta$ would be a regulator $R$ that is computing a function $\tilde{f}(d)$ that is very close to the real $f$. Looking at machines this way, computation becomes the balancing between an input and its corresponding output state. Since $T$ must only evaluate the correctness of the output, it can often be implemented efficiently: Solutions to computational problems in the class $NP$ can still be evaluated in polynomial time.

### 3.3.1 Construction

We will now consider the process of construction. In cybernetics, construction of a particular structure is equivalent to the selection of a particular structure (Figure 3.7). In this abstraction $R$ is the set of all all possible regulators of the world and $C$ performs a selection of the particular one to be constructed. The signal that $C$ sends to $R$ must be unique selection of a particular regulator $R'$ and is therefore the complete description. The dashed lines of the figure depict feedback from the world on the construction process. A construction process can monitor the value of the essential variables of the system and decide to refine the choice of regulator. Also, because $C$ receives information about the disturbances, it could include this information for the correct selection of $R$. This is essentially what is being done in the prototyping and testing phase of an engineering project. Because the constructor is not part of the built system itself, it can not refine the choice on the fly but must first select it, then expose it to the world and then monitor its performance before refinement. This happens on a much longer timescale than the process of regulation.

A self-constructing system (Figure 3.7 right) incorporates the constructor inside the system to be constructed. By constantly monitoring the disturbances and the set of essential variables, a self-constructor can refine its choice in real-time.
Figure 3.7: **Left:** Construction of a system can be viewed as the selection of a particular regulator $R$. The selection is carried out by some constructor $C$. The dashed line depicts a trial-and-error phase of design and engineering. If the selected regulator does not manage to keep the system within its acceptable states, the construction has to refine its selection based on the essential variables of the system and the disturbances it was subjected to. The process of refinement is happening on a much longer timescale than the process of regulation. **Right:** A self-constructing system. The dashed lines depict the border of the self-constructing system. Because selection of the regulation is incorporated in the system itself, the influences from the disturbances and the essential variables are incorporated and reacted upon on a much faster time scale.

Figure 3.8: A symbolizer. The constructor $C$ not only selects the regulator $R$, but also a *symbolizer* $S$. The task of $S$ will be to only promote relevant information for the regulation of $T$. 
3.3.2 Symbolizing and Learning

In most cases, a behaving system allows for abstraction or symbolizing of input disturbances. Stated in terms of cybernetics, this means that the input allows a reduction of variety that does not alter the behavior of the system reflected by its output on the essential variables $E$. Systems allow a reduction of input variety, if the action of $D$ on $T$ can be grouped into equivalence classes of states that result in equal output. An example of such an abstraction can be seen in a system that might want to predict the behavior of a falling object. For most practical purposes, such a system does not need to perceive the trajectory of every single atom in the falling object, but only needs to encode the center of gravity of the compound aggregation of atoms. The particular configuration of atoms does not influence the trajectory of the object. In such a system, the symbolizer would be a mapping from the set of perceived atoms to their center of gravity which is clearly a reduction of parameters and thus variety. Generally formalized, a system is symbolizable, if it allows a mapping $S$ and an alternate description of the world $T'$ such that the following diagram commutes, i.e. the output remains the same independent of which way the information flows ($D \rightarrow D' \rightarrow E$ or $D \rightarrow E \rightarrow E$).

$$
\begin{array}{c}
D \xrightarrow{T} E \\
\downarrow S \quad \downarrow \text{id} \\
D' \xrightarrow{T'} E
\end{array}
$$

(D acts on $T$ in equivalence classes through the equivalence class projection $S$. The benefit of such systems is that a regulator $R$ must now only regulate to block the output variety of $S$ and not that of $D$. By the law of requisite variety it can do so by passing on only $S$'s output variety, instead of the world's ($D$'s) complete variety. Of course it is much easier to construct (or select) a regulator that blocks a smaller variety. Another interpretation of $S$ can also be a perceptive organ like a visual system. Our visual system blocks the bulk of the environmental variety (for example all the electromagnetic waves not in the visual spectrum) because it is not important for us to consider in most of the regulation processes (behaviors) we do.

Even though $S'$ is obscured by the mechanisms of the world and might not be accessible by a constructor $C$, it can be retrieved from observation of $D$ and sometimes also $E$. Retrieving $S$ just from observation of $D$ would be an unsupervised feature detector, while $E$ can act as supervisor and enhance
the feature detection. The task of a learning self-constructor $C$ is now to select the optimal symbolizer $S$ that is ideally equivalent to the real $S'$ of the world (Figure 3.8). $S$ symbolizes the world while $R$ is now a model of the world in terms of the symbols provided by $S$. In Chapter 6 we will see an example of such symbolizer.

3.4 Conclusion

Von Neumann laid out the principles of self-construction some fifty years ago. His demonstration of self-construction by means of an explicit description code, a universal constructor and a code-copier still holds as prime example of self-construction. On the other hand, we have seen that self-construction according to a blueprint is only one part. As observed in nature, the core benefit of self-construction is rather the self-maintenance and the possibility of learning throughout development.

The idea is to link the construction process into feedback loops with the built structure at several scales. A locally maintaining construction loop monitors the micro structure and builds and repairs it while a global construction loop utilizes emergent phenomena such as pattern formation in structure or behavior in function to guide the further development. Because the system is developing from the inside out already embedded in its environment, information from the environment can easily be incorporated into the structure. We saw that it makes sense for a system to narrow its input through a symbolizer to incorporate only input that is of relevance to it. Such a symbolizer can optimally be constructed from observation of the world and does not need to be contained in the description code.

However, employing the principle of feedback in construction is tricky because the rules of construction are not as easy to write down as an explicit description. Such an implicit description requires a deep understanding of how function should affect structure and how global properties can be read and interpreted locally to guide further development.
Chapter 4

A Self-Constructing Artificial Organism

4.1 Introduction

In this chapter we present a model that employs self-construction with the traits discussed in the earlier chapters. The rules of the game are described and it will become clear how, in principle, self-construction could work. We write an explicit Description for the example of a simple multi-cellular organism that expresses attractive or aversive foraging behavior analogous to a Braitenberg vehicle [21]. We demonstrate its self-construction from a single cell, measure its capability to self-repair, and evaluate its behavioral performance.

Our Description describes a set of elementary intracellular mechanisms (factories) that support the self-construction of simple structures such as cell aggregations of specific sizes or segmentation of populations into two regions. These mechanism can be used in a hierarchical ruleset to specify finer grained structures. Branches of the ruleset tree represent sub-structures, which can be designed independently of the rest of the system. Through incremental application of the rules, a high degree of organismal complexity can be theoretically achieved[126].

Our multicellular construction process depends on local feedback of the unfolding structure to guide overall development. Much like in the development of natural organisms, the localized construction process is a function of its surrounding structure and the description residing in every cell. By writing morphogenic messages on their environment, the individual cells contribute to global morphogenic signals that other cells can use to modify their local developmental actions. This coupling can lead to global
Overall, our self-constructing process can be seen as follows. The final organism is encoded in the Description of the 'stem cell' as a kind of state machine in which the states, once activated, become persistent. Each state corresponds to a population of cells that has a particular functionality by virtue of the particular set of elementary intracellular mechanisms that it expresses. Transitions between the states (and so the development of later cell populations) are triggered by local environmental conditions, which are themselves a function of which states have previously been activated.

Figure 4.1 (on the next page): The cell, the Description and the physical instantiation of the Description. a Schematic of a cell and its most important components: a Description code, and Factories. The Description is analogous to DNA, it consists of genes coding for specific sets of factories. Genes have control regions that encode the conditions required for their activation or suppression (small polygons denote intracellular factors. The conditions they match are indicated by a shaded square). Activated genes enable the construction of factories ($B_0$ in the figure). There are three types of factories: the Constructor, which reads active genes and builds other factories; Reactor factories, which act on intracellular morphogen concentrations; and Competence factories, which implement cellular competences such as migration. Each gene corresponds to a particular cell type, and so the same symbol denotes both the gene and its cell-type ($B_0$, $C$, etc.). b The cell lineage tree: Nodes denote cell types and edges the lineage paths between the types. Blast cells lie at the root of the tree, and specialization occurs centrifugally toward the fully differentiated leaves. Passage along the edges is controlled by the concentrations of intracellular factors and diffusible morphogens. The cell in the figure is expressing factor $B_0$, which is one condition for activating gene $C$. If, in addition, certain environmental conditions are also met (code in supplementary material), then daughters of $B_0$ will follow the path from $B_0$ to $C$. c The body plan of the physical instantiation that is laid out by the Description: Cells of the same types cluster together and build functional entities. The blast cells $B_\ast$ form smaller populations and create and maintain their functional cell populations (for example $B_S$ cells give rise to the $S$ cell population).
4.2. The Model

![Diagram of the Model]

- **a**
  - Diagram labeled as 'Cell', 'Constructor', and 'Competence'.

- **b**
  - Diagram showing nodes labeled as $B_0$, $C$, $S$, $B_S$, $B_M$, and $M$ with arrows indicating interactions.

- **c**
  - Diagram showing the relationships between $S$, $C$, and $M$ with $B_S$, $B_0$, and $B_M$ as intermediates.
4.2 The Model

4.2.1 The Cell

The basic building block of our multicellular organism is a cell (Figure 4.1a). Each cell is equipped with a Description, and a Constructor. The Description is composed of genes that encode explicit specifications of individual intracellular factories, but do not explicitly specify the entire organism. The genes are activated by the local concentration of morphogens. Activation permits the Constructor to read the gene, and so to construct the specific factories that correspond to that gene.

The cells interact with the environment through chemical diffusion of morphogens. The coefficients of diffusion through the membrane for the different chemicals are constant. Because we have chosen diffusion through the membrane to be passive in our model, the cells have no active control over which chemicals enter their interior from the environment. Once a particular cell is formed by replication, its further dynamics are determined entirely by its internal chemical concentrations and the functions of its expressed factories. They interact with the environment solely through their reactor dynamics. It is important to note that individual cells can not perform complicated algorithm-like computations, nor can they communicate globally. It is only the interplay of the local dynamics of all the cells in the growing population that gives rise to the non-trivial structure of the organism.

The Description Code

The code specifying the rules for the construction of the organism is identical in each cell of the same organism. When the cells divide, an exact copy of the code is made and placed into the daughter cell. The Description encodes:

- The definition of the lineage tree in the form of a differentiation graph that encodes the chemical conditions under which genes become active.
- Which factories the constructor should build depending on the cell type.
- The specification of the factories (competences and reactors).

The structure of the organism is completely specified by these local criteria and depends on them. Because the code is identical in all the cells,
4.2. The Model

the differentiation mechanism has to activate and deactivate parts of it, depending on the cell type. In principle, other types of organisms could be built by just applying another code.

The code defines local objectives and actions in order to fulfill those objectives. We do not specify the explicit structure of the grown organism but only describe what a single cell must achieve in order for the whole organism to assemble to the target structure. Because a cell can only act locally the construction of the organism relies on global cooperative phenomena like those specified in section 4.3.1.

Modular Components within the Cell (Factories)

The cells can contain a set of modular functional components (factories) (Figure 4.1a). All factories are described in detail in Section 4.3.

- Constructor factory: This factory initially reads the active gene in the Description code and builds the corresponding factories associated with the active gene. The Constructor factory is cloned into a newly born cell at cell division.

- Chemical reactors: These implement the intracellular mechanisms involved in the production or consumption of morphogens. They set up the global morphogenic gradients across the organism (AxisReactor, InterAxisReactor), and are also responsible for the production of morphogens (SourceReactor, ConstReactor) that are used for the maintenance of the preferred population size as well as intracellular signals.

- Competences: These express physical capabilities of cells such as chemotaxis (MigrateCompetence), implementation of sensory or motor modalities (SensorCompetence and MotorCompetence), axonal growth (AxonCompetence) and cell division (DivideCompetence).

The activation of these cellular mechanisms depends on the description of the system, and the environmental context in which the cell resides. The central principle of our construction method is the appropriate coordinated instantiation of reactor factories. By contrast, the competence factories are merely convenient stubs, whose detailed implementation is not directly relevant to the process of self-construction that we describe in this article. Thus, in our example organism below, the MotorCompetence and SensorCompetence implement physical properties that give rise to the behavior of the final organism and are not directly involved in the self-construction and self-repair of the organism.
Differentiation Scheme

Differentiation of a cell is governed by its local environment, its Description code and its history. The Description code is analogous to real DNA. It consists of genes that code for specific sets of intracellular factories. As in nature, the genes have control regions that encode the conditions for their activation or suppression \([63, 76, 28]\). These chemical conditions are simple thresholds that specify whether the gene is switched on or off (see table B.2). If all its conditions are met, a gene is activated. The encoding regions contain the description of the factories to be instantiated. A newly born cell is initially undifferentiated, but inherits the chemical configuration of its mother cell. If a gene of the undifferentiated cell becomes active, the Constructor reads its encoding region and constructs the required factories, so differentiating the cell. Once differentiated, the cell remains locked to its type because it is defined by the intracellular factories it has constructed.

The conditions under which a gene is activated are programmed in the Description to reflect the developmental need for a cell of a specific type. These conditions might only become true in a cell that is born at a specific moment in development and at a particular position.

Genes can also be regulated by chemicals that do not diffuse across the cell membrane. These chemicals are released from reactors within the cell. Because non-diffusible chemicals are local to the cell, they represent the state or the history of this cell. Non-diffusible chemicals are denoted by a fraktur font \((B_0, B_M, B_S)\). These chemicals enable the activation of a specific gene in an offspring of the cell because they, like all other chemicals, remain present in the interiors of daughter cells after cell division. This inheritance mechanism gives rise to a lineage tree (Figure 4.1b) where only a subset of cell types can arise from a given mother cell. The offspring subset of cell types for a given mother cell type expresses a requirement condition for the non-diffusible marker chemical of their mother cell type. Thus, the development of the entire organism from a single progenitor cell can be represented as a lineage tree in which the nodes denote cell types and the edges denote the developmental paths between the types. Specialization occurs centrifugally along edges of the lineage tree, allowing sub-structures on one branch to develop independently of those on other branches. For example, cells derived from the sensory blast cells, \(B_S\), will setup and maintain the sensory system autonomously and independently of other subsystems (Figure 4.1c).

Cell replication occurs as follows: As described above, a gene whose conditions in the regulatory region are active, expresses a developmental need for a cell of this type. Cells containing DivideCompetences will now
become active. The DivideCompetence will sense active genes within its cell and consequently initiate a cell division. During a cell division, the chemical configuration of the mother cell is cloned inside the newly created cell. The mother cell remains of the same type, while the daughter cell is at first in an undifferentiated state. However, being subject to the same internal and external chemical conditions as its mother cell, the gene that triggered the cell division becomes active also in the daughter cell. The Constructor factory of the daughter cell now reads the activated gene and instantiates the corresponding factories. It thus becomes of the expressed type which is, by virtue of the non-diffusible markers, necessarily the same type or a subtype of the mother cell.

4.2.2 The Environments

The world in which the system is embedded is divided for convenience into two environments of different scale (Figure 4.2). The local environment is a two dimensional lattice whose size is on the order of the organism’s diameter and is used to model the immediate physical environment of the organism. The developing cells are located at the nodes of the lattice. Passive diffusion of morphogens occurs along the edges of the lattice, and also from the nodes into the interior of the cells. The natural neighborhood relation on the lattice provides the basis for cell adhesion and local migration of cells. Individual cells can only sense other cells located at neighboring sites, and they can only migrate along the edges of the lattice. In principle, multiple cells can be co-located on top of one lattice node. But, if they are of the same type, they are very likely to move to a free neighboring node because of constraints they must optimize (see Section 4.3.2).

Various morphogens diffuse along the edges of the lattice. Each chemical species has its own diffusion coefficient. Diffusion is governed by the discretized diffusion equation:

$$c_i^r \leftarrow c_i^r + \tau \cdot D_i \cdot \sum_{r \sim r'} (c_i^{r'} - c_i^r), \quad (4.1)$$

where $c_i^r$ is the concentration of chemical $i$ at location $r$ and $r \sim r'$ holds true for neighboring nodes $r$ and $r'$. $D_i$ is the diffusion coefficient for chemical $i$ and $\tau$ is a small integration constant.

The 2D diffusing grid is itself embedded in the second, larger scale world environment. Goods are scattered in this larger world. Cells that express the SensorCompetence (see Section 4.3.2) are able to sense the concentration of good at their location in the greater environment. The local environment (and so the whole population of cells) is moved with respect
Figure 4.2: The two environments of the simulation. The cells reside on nodes of the lattice in the local environment. The local environment itself is embedded in the world environment, so that each cell receives a physical location in the world environment. Forces applied to the local environment result in movement and rotation of the local environment with respect to the world.

to the greater world environment by means of those cells that express the MotorCompetences. The MotorCompetences apply forces to the local environment. Together, they result in a single forward force $f = \sum_i f_i$, and a torque $M = \sum_i M_i$, which is applied to the center of gravity of the cell population. The translation $\mathbf{x}$ and rotation $\theta$ of the local environment is then computed by the following equations that approximate dynamics of a rigid body with strong friction:

\begin{align*}
    m \cdot \ddot{\mathbf{x}} &= f \\
    I \cdot \dot{\theta} &= M.
\end{align*}  

$m$ and $I$ are appropriately chosen constants.

### 4.3 The modular factories

The following factories are used by the developmental process. The mechanisms offered by the factories are quite general. In this section we will describe them and how they are used in our example organism.
4.3.1 Reactor factories

The reactor factories implement a set of differential equations that express the cellular chemodynamics.

**ConstReactor**

This reactor maintains the concentration of a specific chemical $c_i$ at $\hat{c}$ by the production rule

$$\dot{c}_i = \alpha \cdot (\hat{c} - c_i).$$  \hspace{1cm} (4.4)

In our model we have used this reactor to maintain the concentration of a non-diffusible marker chemical that characterizes its cell type. Every cell contains a reactor of this type.

**SourceReactor**

The SourceReactor produces chemical $c_i$ at a constant rate $r$:

$$\dot{c}_i = r.$$  \hspace{1cm} (4.5)

In our model, we have used SourceReactors to maintain the sizes of the various cell populations. Within a population, all cells release their characteristic diffusible marker at a constant rate. Because the volume of any cell population grows more rapidly than its enclosing surface, there will come a time when the overall production of the marker within the population exceeds the overall outwards diffusion of the marker across the enclosing surface of the population. At this time, the concentration of the marker within individual cells begins to rise and this signal is used to inhibit further cell division.

**DecayReactor**

A DecayReactor decays the concentration of chemicals $c_k$ through $c_l$ at rate $r$:

$$\dot{c}_i = -r, \quad \forall i \in \{k, \ldots, l\},$$  \hspace{1cm} (4.6)

where $k, l$ and $r$ are configurable parameters. It does exactly the same as the SourceReactor, but with negative rate $r$. We have included the DecayReactor for convenience: In the XML code it allows the specification of decays for several chemicals with same decay rates in one single factory (see B.1).
AxisReactor

AxisReactors are used to establish the spatial organization of cell populations (See Figure 4.3a). They do so by creating gradients of morphogens whose concentrations provide signals for the conditionally activation of various processes in other cells.

AxisReactors are based on rate equations proposed by Gierer and Meinhardt [80]. A given AxisReactor generates chemical dynamics within each cell that lead to the formation of opposing gradients of two morphogens $g_1$ and $g_2$ across an entire population of cells. These gradients provide an axis for development. The gradients arise by a cooperative-competitive process. Local competition between the production of morphogens $g_1$ and $g_2$, and long-range cross-facilitation of their production by two additional signal chemicals $s_1$ and $s_2$. This process follows the dynamics:

$$\dot{g}_1 = \frac{c s_2}{a + g_2^3} - \alpha g_1$$

$$\dot{g}_2 = \frac{c s_1}{a + g_1^3} - \alpha g_2$$

$$\dot{s}_1 = \gamma g_1 - \gamma s_1$$

$$\dot{s}_2 = \gamma g_2 - \gamma s_2.$$  

Where $c$ is the gain of the facilitation, $\alpha$ is a temporal decay, $\gamma$ is the gain for the production of the cross-facilitators and $a$ is an arbitrary constant.

Populations of cells employing the following chemodynamics will stabilize into a two state structure along its longitudinal axis where the states are characterized by high $g_i$ and low $g_j$ concentrations for $i \neq j$.

The diffusion coefficient for the long-range cross-facilitators $s_1$ and $s_2$ must be higher than that of the short-range competitive chemicals $g_1$ and $g_2$. $g_1$ gives rise to the production of $s_1$ which in turn will cross-facilitate $g_2$ over a longer distance and vice versa for $g_2$ and $s_1$. The slow diffusion of the $g$s will influence neighboring cells to belong to the same cluster, whereas rapidly diffusing long-range facilitation chemicals $s_1$ and $s_2$ break homogeneity along the axis.

InterAxisReactor

InterAxisReactors are used to create spatial organizations with respect to already established gradients. To do this, the InterAxisReactors are similar to the AxisReactors but contain an interaction term in the production rule of $g_1$ and $g_2$. 

4.3. The modular factories

\[ g_1 = \frac{c s_2 (\delta + g_1^I)}{a + g_2^3 - \alpha g_1} \quad (4.11) \]
\[ g_2 = \frac{c s_1 (\delta + g_2^I)}{a + g_1^3 - \alpha g_2} \quad (4.12) \]
\[ s_1 = \gamma g_1 - \gamma s_1 \quad (4.13) \]
\[ s_2 = \gamma g_2 - \gamma s_2. \quad (4.14) \]

\(g_1^I\) and \(g_2^I\) are the chemical concentrations of the interacting morphogen, \(\delta\) is the interaction baseline. If no interaction morphogens are present, these dynamics coincide with the dynamics of the AxisReactor with a facilitation gain \(c \cdot \delta\). \(\delta\) should be tuned such that the gain is not high enough for segmentation to occur when there is no influence from \(g_1^I\) and \(g_2^I\). This way we can ensure this segmentation to wait until the influencing gradient has been set up.

In our example the InterAxisReactor is instantiated by the sensory cells and establish an anti-parallel gradient with respect to the gradient across the motor cells (see Figure 4.3b). This anti-parallel structure is later used to support the crossed sensorimotor connectivity of the growing axons.

### 4.3.2 Competence factories

The cells can express predefined competences. Like the chemical reactors, the competences are built by the Constructor when the cell differentiates.

**DivideCompetence**

If a cell expresses the DivideCompetence, it can divide asymmetrically into two cells. However, this division occurs only if the differentiation graph has an active edge signaling an environmental need for a certain type of cell (i.e. a gene becomes active). Division will occur with a predefined low probability, which keeps the production rate low. After a binary cell division, one daughter cell remains the mother cell while the other daughter cell will undergo differentiation beginning from the state of its mother and then following the differentiation graph as discussed in Section 4.2.1. This daughter cell can remain of same type, or become a more specialized type.
A Self-Constructing Artificial Organism

Figure 4.3: The AxisReactor and InterAxisReactor create a global gradient across the population they are instantiated in. a The AxisReactor is instantiated in all the $C$ cells and thus creates a gradient across the complete organism. This two-state structure is used to determine front and back of the organism. b The InterAxisReactor is instantiated in the sensory cells and uses the influence of the motor cells to determine its direction.

**MigrateCompetence**

Cells that express the MigrateCompetence can change their location on the grid. They attempt to optimize their positional objective function which is calculated from the number of neighbors and the concentration of chemicals in the environment. A cell will try to maximize the number of its neighbors, while attempting to reside alone on a grid point. A term that depends on the chemicals in the surrounding environment encourages the cell to migrate according to its chemical affinity. The Description code describes which chemicals a cell of a certain type is attracted to and which chemicals repel it.

Each cell $p$ attempts to minimize its free energy

$$H_r = \sum_{r' \sim r} J(s_r, s_{r'}) + S(s_r) + J_p(c_r),$$

where $r$ is the location of the cell $p$. $J(s_r, s_{r'})$ describes the binding energy of two neighboring nodes at $r$ and $r'$ and may take the values $-\epsilon$ if both sites are occupied by at least one cell or 0 otherwise. $S(s_r) = \lambda \cdot ([# \text{cells of same type at } r] - 1)$ expresses the stress of a cell at node $r$. $\lambda$ is large compared to $\epsilon$, so that cells of the same type compete for a space
on a particular node. $J_p(e_r)$ expresses the affinity of a cell for the node’s morphogen configuration. $J_p$ depends on the cell type and is configured by the MigrateCompetence (specified in the Description, see Appendix B.2).

A migrational step of a cell is performed according to a Monte Carlo algorithm similar to the cell sorting algorithm employed by Marée et al.[77]: The cells arrange themselves so as to minimize the free energy of the whole cell population through local optimization. At each active migrational step, a cell picks a random neighboring target node on the lattice and calculates the energy difference $\Delta H$ between the current configuration and the configuration after a hypothetical migrational step to the chosen node. The new configuration is accepted if $\Delta H \leq 0$. If $\Delta H > 0$ the new configuration is accepted with probability $P = \exp(-\Delta H/T)$. $T$ is the temperature of the process and measures the degree of migrational fluctuations of the cells. $T$ is kept constant throughout the course of a simulation. This will result in a fluctuating equilibrium structure. We chose not to use annealing, so that the system is able to react quickly to perturbations and to reorganize if necessary, without having to modulate individual temperature parameters.

**AxonCompetence**

In order to achieve macroscopic behavior, the growing organism needs to build functional components. To support function, axons can signal neural activity of their originating cell to remote cells. A cell expressing the AxonCompetence grows axons constrained by attraction and repulsion to specific morphogens. Axons form synapses with the cells to which they are connected, and so transfer their neural activity to their post-synaptic cell. The growth of the axons is guided by the same mechanism as the migration of the cell through the MigrateCompetence. A growth cone at the tip of the axon optimizes its objective function by migration according to a Monte Carlo process. As specified by the code, the objective function encodes the optimal chemical environment and includes a competition amongst axon terminals on the same postsynaptic cell. This way, axon terminals will distribute homogeneously within the target environment. During the migration of the growth cone, the axon grows or retracts accordingly.

**SensorCompetence**

The SensorCompetence senses the concentration of ‘good’ at its cell in the world environment. The concentration produces a proportional cellular activity that can be transmitted via an axon.
MotorCompetence

The MotorCompetence applies a force proportional to the cells activity to the whole organism (see Section 4.2.2). The direction of the force with respect to the body orientation is specified in terms of chemical gradients and read from the Description code. The morphogens of the global positional gradient (AxisReactor) are used to determine the direction of the force.

The morphogenesis of the actual muscle or effector that forces a movement of the organism is omitted in our model for simplicity. It is important to note, however, that in principle this morphogenesis could be simulated. We have a clearly identified region of the organism that is responsible for movement and the specialized motor stem cells could be programmed to further divide into more specific cells that compose an effector.

4.4 Design and Development

The potential of our developmental scheme can be demonstrated by the self-construction and self-repair of a simple, but non-trivial behaving functional organism. The final structure and function of this organism is similar to 'vehicle' number 2b (aggression) proposed by Braitenberg [21]. Figure 4.4 shows the template structure from Braitenberg. It contains a sensory input population and motor output population. Excitatory axons connect the left side of the sensory map to the the right motor cells, and vice versa. This connectivity yields an organism capable of targeting and seeking objects of sensory relevance in the environment.

4.4.1 Design of the Code

The final structure of the self-constructing organism is shown in figure 4.1c. This structure is reflected in the Description code (Figure 4.1b) in the following way: each gene corresponds to a cell type, and each cell type is responsible for the setup and maintenance of a particular substructure of cells. In our example, we call cells that contain a DivideCompetence blast cells. They owe their name to their biological counterparts which are incompletely differentiated progenitor cells that give rise to differentiated cells through asymmetric division. The skeleton of the hierarchical code consists of a tree of blast cells ($B_0$, $B_M$ and $B_S$). A blast cell constructs its appropriate substructure by generating a population of differentiated cells. $B_0$ are the topmost blast cells. They give rise to the population of cells $C$ that are the scaffolding for the entire organism (Figure 4.1c). By placing
Figure 4.4: A Braitenberg vehicle with excitatory connections. In the front of the vehicle are two sensors and in the back two motors. The lines depict connections. **a** illustrates how aversive behavior is produced by simply connecting the left sensors to the left motors. **b** shows attractive behavior by connecting in a crossing manner. Figure is copied from [21].
AxisReactors (see Section 4.3.1) in the specification of the $C$ cells, a body axis forms within the population of the $C$ cells and divides the organism into front and back. $B_M$ and $B_S$ cells govern the development of the motor and sensory substructures consisting of $M$ and $S$ cells respectively. Their genes encode a MigrateCompetence that causes them to migrate to their corresponding extremities of the organism (motors in the back and sensors in the front). $M$ and $S$ cells only arise in an environment that has already been prepared by the higher level structure created by $B_0$ and $C$ cells. The $M$ cells contain AxisReactors to further divide into a left and a right subpopulation. The $S$ cells contain an InterAxisReactor to form an antiparallel division to that of the $M$ cells in order for the axons to be able to find the correct motor cells.

Each blast cell and its corresponding differentiated cells maintain their designated substructure of the organism independently. Self-construction and self-repair is therefore localized within these substructures.

For our example of a Braitenberg vehicle, we need two levels in the hierarchy, a global body plan on the top level and a motor and sensory population on the second level. The code is provided in table 4.1 in tabular form. The tree structure is implemented by the intracellular chemicals $B_0$, $B_M$ and $B_S$ and their corresponding activation conditions in the genes.

### 4.4.2 The Description Code

Table 4.1 displays the complete Description Code of the organism. The tree structure of the code is implemented by means of the non-diffusing chemicals $B_0$, $B_M$ and $B_S$. For example: A cell of type $B_0$ includes a ConstReactor for $B_0$. Therefore only the genes $B_0$, $C$, $B_M$ and $B_S$ can be activated in a cell of type $B_0$ and thus, a $B_0$ cell can only divide into cells of type $B_0$, $C$, $B_M$ and $B_S$ but not into more specialized cells of type $M$ or $S$. This reflects the lineage of the code depicted in Figure 4.1 bottom.

The table reads as follows: The second column ($C$) states that if the morphogen concentration of $B_0$ is greater than $\eta$, that of $B_0 > \hat{b}_0$, but the concentration of $c$ is less than $\hat{c}$, the cell differentiates into a cell of type $C$. Once differentiated, the Constructor will instantiate the following factories: A MigrateCompetence with symbolic argument $+c$. Cells with this competence will migrate towards higher concentrations of $c$ in the environment. A ConstReactor with argument $C$ keeping the intracellular concentration of $C$ constant and greater than zero (in blast cells this regulatory factor is used to determine the branch of the lineage for its descendant cells). A SourceReactor for the morphogen $c$ produces $c$ at a constant rate and so
Table 4.1: The Description Code of the organism in tabular form: The upper part of the table describes conditions on morphogens for a particular gene/cell-type to become activated: the expressions in the table denote the comparison of the morphogen concentration to some constant. Once the gene is activated (all conditions are true) and the cell has differentiated, the corresponding factories (lower part of the table) are instantiated. The entries in the lower part are instantiation parameters to the factories and represent symbols for different morphogens. A check mark signifies that this Competence is instantiated. For MigrateCompetence + and − defines whether this cell is attracted or repelled by the corresponding morphogen. The AxonCompetence takes as an argument a list of morphogen to morphogen mappings. A mapping \( a \Rightarrow b \) signifies that a cell with high internal \( a \) concentration sends its axons to locations with high \( b \) concentration. The reactors and competences are explained in Section 4.3.
A Self-Constructing Artificial Organism

signals to neighboring cells its existence and allows them to migrate accordingly. Finally, an AxisReactor that will be responsible for the creation of the $g_1^C, g_2^C$ gradient which in turn will be used to activate the genes $B_M$ and $B_S$ which are conditioned on $g_1^C$ and $g_2^C$.

4.4.3 Development

This section describes the sequence of steps in the development from a single cell to the functional organism. Figure 4.5 illustrates the state of the organism and the state of the genes in a highlighted cell.

Figure 4.5(a): Start of ontogeny, placing a single cell into the environment

The development process begins when a single cell is placed in the environment. This cell contains the complete information needed for the self-construction of the organism. The environment has no specific chemical configuration. According to the graph on the left hand side, the cell will differentiate into further $B_0$ cells (yellow). By virtue of their SourceReactor, these cells release morphogen $b_0$ into the environment.

The threshold $\hat{b}_0$ of condition $B_0$ defines the size of the $B_0$-population. Because the morphogens $b_0$ released by $B_0$-cells’ SourceReactors diffuse away in the environment, the population must reach a critical size in order establish a stable chemical concentration. This size depends on the diffusion coefficient and the production rate of the morphogens. The critical size has the property that the production of chemicals inside the volume of the body is of the same rate as their diffusion away through the body’s surface. A similar mechanism for the determination of size of growing organisms was already noted by D’Arcy Thompson almost hundred years ago [112].

The $B_0$ cells will form an initial population of stem cells, until conditions for $C$ are reached ($b_0 > \hat{b}_0$) and $B_0 \rightarrow B_0$ is switched of. The size of this initial population is not critical. However, the more stem cells an organism has, the faster it will grow. And with more stem cells, an organism is more likely to survive damage. If all stem cells are killed, the organism can not rebuild its primary structure.

Figure 4.5(b): Dividing into positional structure cells $C$

Once the $B_0$ population has been set up and the conditions $C$ have been reached, the $B_0$ cells begin dividing into $C$ cells. The same mechanism that
determined the size of the population of \( B_0 \) cells will limit the size of the population of \( C \) cells: Their SourceReactors produce the marker chemical \( c \) and the \( B_0 \) cells will stop dividing into \( C \) cells when their critical concentration according to the parameter \( \dot{c} \) is reached within the population. As specified in the code, the \( C \) cells instantiate AxisReactors that will create the body axis.

**Figure 4.5(c): Setting up the positional structure**

At this point, the population of \( C \) cells has reached its critical size, and division into further \( C \) cells is switched off.

The AxisReactors in the cells of type \( C \) will set up a body axis by segmenting the population into two equally sized clusters of minimal boundary length (see Section 4.3.1). This segmentation happens as a cooperative effect amongst the population of identical AxisReactors and is not influenced by any external signals, except the noise in the environment to break the initial symmetry. While still composed of type \( C \) cells, the two clusters are defined by their chemical footprints: The back cluster is identified by high concentrations of \( g_{C_1} \) while the front cluster has high concentrations of \( g_{C_2} \).

According to the conditions for \( BM (g_{C_1} > \theta) \) and \( BS (g_{C_2} > \theta) \), when the clusters have completely formed, the \( B_0 \) cells will start to divide into \( BM \) (blue) and \( BS \) (orange) cells depending on their environment. \( BM \) cells are motor neuroblast cells which generate the motor units, while \( BS \) cells are sensory neuroblast cells that will generate the sensory units. The Migrate-Competence allows these specialized blast cells to migrate to the edge of the population driven by repulsion from the opposing chemical (\( BM \) cells are repelled from chemicals released by \( C \) cells in the sensory cluster and vice versa). I.e. motor neuroblast cells migrate to the back while sensor neuroblast cells migrate to the front.

**Figure 4.5(d): Preparing to develop motor and sensory units**

The initial population size of motor blast cells has been reached and the conditions for \( M \) become true. Consequently, the production of motor cells \( M \) (green) has started. The motor and sensory cells will also contain (Inter)AxisReactors and hence form a Left/Right segmentation within their population. Because AxisReactors segment a population along its longer principle axis and the \( M \) population is elongated orthogonal to the body axis, the Left/Right segmentation will be orthogonal to the global asymmetry.
The sensory subpopulation will form a gradient anti-parallel to the motor subpopulation by virtue of the InterAxisReactor: The marker morphogens $g_1^M$ and $g_2^M$ of the motor population influence the orientation of the segmentation of the sensory subpopulation into $g_1^S$ and $g_2^S$ (see Section 4.3.1).

**Figure 4.5(e): The developed organism**

Once all populations have been established, all genes become inactive and no further cell divisions take place. The AxonCompetences in the sensory cells grow axons to reach the motor cells by a simple wiring strategy described in the code: they following the corresponding marker morphogen – yellow sensory cells target yellow motor cells and vice versa (see Section 4.3.2)

The structure of the organism is now in stable equilibrium in the sense that the blast cells have all become quiescent and no longer divide, and all axons have grown to their target. The organism has now finished its development and is behaving as expected (Figure 4.7).

### 4.5 Behavior and Self-repair

The development leads to an agglomeration of different cell types that are organized spatially in a configuration that can support a behavioral function. In this case, the organism’s minimal sensorimotor organization is similar to Braitenberg’s classical ‘aggression’ vehicle [21], which is attracted to a stimulus such as light or food. We chose a simple task, tracking ‘goods’, to evaluate the behavioral competence of the mature organism (Figure 4.7). The sensory cells transduce the local concentration of goods into neuronal...
4.5. Behavior and Self-repair
activity. This signal propagates along the axon and excites the post-synaptic motor cell. Excited motor cells apply a force in the direction of the longitudinal morphogen gradient of the organism, resulting in a momentum and torque of the compound population.

Figure 4.6 shows the first twelve examples of simulated organism. From the point of view of development, all twelve organisms have developed properly to the expected structure: Front and back are clearly separated, the motor and sensory cell populations are present and well sized, the axons grow from front to back and connect in a crossing manner to the motor cells. Despite the nicely set up structures, their corresponding behaviors are not always convincing. This is due to the fact, that the behavior is delicately sensitive to the exact structure. A single motor cell that is placed a bit too far out from the center can create a spiraling behavior. But as shown in Appendix A, even perfectly developed organism give rise to oscillating behavior because of the dynamics of the Braitenberg vehicles. The quality of behavior is therefore not a very benevolent measure for the performance of the development.

The distributed construction process not only drives the development of the organism but also permits it to repair itself. When the organism is damaged, the quiescent blast cells near the site of injury sense changes in the concentrations of the morphogens of the destroyed cells. The differentiation conditions in the nearby blast cells will be reactivated and the cell division and differentiation process will restore the local structure. This process occurs only at locations where damage is and continues only until the cell populations has been restored to their original size and organization. When the equilibrium state is reached, the blast cells become quiescent again. It is this natural reactivation of quiescent blast cells that allows the robust development and repair.

The performance of goods-tracking was measured as the mean efficiency of the organism, given by the ratio between distance traveled and goods encountered in a trial of 30,000 steps (Figure 4.7b). Well developed organisms transform the environmentally mediated sensory activity into an overall coherent forward movement along the path. After a development period (age 120,000 steps), we damaged the organism by removing a circular patch of cells with radius of about 4 cells. This damage was applied in a random location in the periphery of the organism, destroying a significant fraction of the sensory or motor region. Damaged organisms are unable to control their movement and hence, their performance is significantly worse than that of the mature organism. However, the organism has repaired itself after 70,000 steps, and its foraging performance has recovered. A movie of the development, behavior and repair of a sample
Figure 4.6: The first 12 simulated organisms and their behavior in the environment
A Self-Constructing Artificial Organism

Figure 4.7: Performance of the organism, and its recovery following damage. **a, Left.** (Top): A fully developed organism in its local environment, at the end of a development sequence (age 120,000 steps). Color codes, as for Figure 4.5. (Bottom): The organism in its greater environment. The local environment is indicated by the small square. The red side of the square corresponds to the upper border of the local environment. The path of the organism is indicated by white dots at equal time intervals. The simple sensorimotor organization of the organism allows behavior: The organism is able to follow the good track (green) for an extended amount of time. **a, Middle.** The damaged organism after removal of a patch of cells: Imbalance in connectivity degrades the performance. Some genes become again active and replenish the damaged cells and, by the same principles of development described above, the structure and function of the organism is restored. **a, Right.** The organism after self-repair (70,000 steps after damage). All the local conditions are again satisfied. The various cell populations and their organizations have been restored. Performance recovers. **b** Mean and standard deviation of the efficiency (Distance traveled / goods gathered in 30,000 steps) for 200 simulated organisms. After damage, the organisms recover.
organism can be downloaded from ref [1].

4.6 Conclusion

We have described a method for the design of self-constructing and self-repairing organisms. This system is able to develop from a single cell into a multicellular organism with nontrivial structure: it is capable of performing the high-level behavior of line following. Here we should note, that our system assembled itself to this functionality without the use of any higher level modules that incorporate complicated function. Rather, it developed a scaffolding and correctly placed motor and sensor cells and the connections thereof by proper migration of cells that are exhibiting simple functions.

Control of pattern formation is crucial to the design of self-assembly. The formation of global patterns in biological systems has been explained by means of reaction-diffusion systems[114, 47] that use only local interactions. A similar phenomenon has been observed in the behavior of social insects[51, 18]. Grassé described how individual animals can use structures previously built by the collective society to guide their individual further construction actions, rather than following simple sequential recipes. He coined this behavior stigmergy: “The worker does not guide his work, he is directed by it”[58]. Previously, the concept of stigmergy has been applied mostly to multi-agent systems in the sense that the agents alter their environment which in turn feeds back onto their behavior[111, 58]. We use stigmergy in the sense that the whole cell population is the environment of the cell, and the local structure of the population guides the action of every single cell. Our genes can be regarded as stigmergic rules. The fully developed organism is the equilibrium state of this decentralized construction process. Because it is a stable equilibrium, perturbations of the population structure will be restored through local actions, so providing self-repair. As discussed in the previous chapters, the organization maintains itself by homeostasis, a property that it shares with all living systems[78].

Self-repair and regeneration of tissue in multicellular systems has been described by Furusawa and Kaneko[44] and Miller[81, 69] in simulations of cellular models. However, neither of those studies explain what the essential components of self-repair are, nor how this property should be implemented in the genetic code, as we have done here. Furthermore, Miller’s system can not down-regulate its growth process, and so it grows ad infinitum because it is only partially sensitive to its larger structure.

Von Neumann conceived system construction as direct feed-forward translation of a Description contained in the parent into a daughter instance
by the constructor machinery. More recent studies of self-construction, such as those reviewed in the Introduction of this thesis, have also taken inspiration from development in biology, which depends on feedback between the expression of genetic description and the environment. Because of the perceived complexity of this process those authors have relied on evolutionary algorithms to find a suitable instruction code. Here, we demonstrated that the biological-style development can be programmed explicitly. That is, the use of genetic algorithms could be replaced by principled design, while preserving the environmental sensitivity of the construction process. To achieve this, we modularized the elementary construction processes in such a way that they can be conveniently composed under genetic control. In particular, we have demonstrated how to exploit cooperative organizational phenomena by explicit description to achieve a global target structure that is capable of a behavioral function. In addition to its self-construction, our system is able to repair itself by the very nature of its construction: The self-construction and self-repair are equivalent mechanisms. The principle components necessary for the presented scheme of self-construction are: (1) A physical cell capable of self-reproduction; (2) a differentiation scheme that is influenced by the history of the cell and its local environmental conditions and (3) modular components that are conditionally activated and can alter the immediate environment of the cell and thus feed back to the differentiation process. It is sufficient that the cells can communicate only by means of passive diffusion of chemical signals. In our model, we have been careful to utilize only local criteria and actions that are in principle realizable in physical systems.

In its present form, our construction process is not universal in the sense that it can construct any arbitrary structure. As for natural systems, it is very unlikely that any arbitrary artifact can be designed to self-construct. The developmental process and its corresponding description on the 'gene' code constrains the space of possible stable functional configurations. However, we introduce a general mechanism for designing structures that could allow a variety of different target structures. A finer grained structure than the one shown here can be obtained by further subdividing regions by means of AxisReactors and by providing blast cell types lower in the hierarchy that control the finer grained development superimposed on the coarser grained structure established by the earlier development process.

In contrast to traditional external fabrication, self-construction does not employ a global external observer to supervise the assembly process. Nonetheless, we suggest, that it is necessary that local processes are sensitive to the global state of the system. We achieve this interaction through cooperative effects like symmetry breaking within the population and a
morphogen sensing mechanism that regulates the size of the population. These locally generated signals arising from global emergent structures reflect qualities of the whole structure and make them accessible to localized processes. Hence, they can be utilized to drive the local development.

Furthermore, we propose a hierarchical structure of the cell-type lineage, such that substructures can grow and maintain themselves independently of the rest of the system, while utilizing structures already set up by earlier processes (higher up in the hierarchy). This is a powerful concept that allows for directed engineering and refinement of these components, while leaving their self-construction and maintenance their own responsibility. This orthogonality of subsystems provides a modularity that facilitates development and repair. In evolving systems, independence between developmental modules also allows more robust evolution by local refinement of subsystems [117].
Chapter 5

Learning in Development

5.1 Introduction

In the previous chapter, we saw how a behaving organism can self-construct starting from a single cell. All the information about its structure was implicitly stored as constructive rules in the Description code. Here, we explore how incomplete information in the Description can be completed by information retrieved from the environment of the developing system. As already mentioned in Section 1.2, we can not learn any arbitrary objective from scratch because the search space is much too large. We need to narrow the domains of interactions in which learning takes place by bootstrapping an initial structure.

As basis for learning we chose the organism of the last chapter, whose code contains a sensory-motor structure but initially no axonal growth preferences. The learning task is for the motors to attract the right axons to get the desired behavior. Like in the completely specified species of the last chapter, sensory axons carry a signature that identifies them as being from one (red) or the other side (blue) of the sensory population. They acquire their identity through the position of their cell body in the gradient across the sensory population. The motor cells can attract a particular axon by releasing a ‘red’ or ‘blue’ attracting morphogen. It is their goal to learn which axons to attract.

We tested two different behaviors: Love and Fear. In the Love objective, the organism is rewarded when spending time on the good and punished for being off the good. Contrariwise, in the Fear objective, the organism is rewarded for spending time off the good and punished while traversing the good. The reward signal is a single global scalar signal presented to all the motor cells. In biology such a signal would be implemented by a fast
diffusing neurotransmitter such as dopamine\cite{102}. Based on their current strategy and the corresponding current reward signal the motor cells have to decide whether to maintain or change their strategy.

The cells can be seen as individual agents that have only a local view of the world. Their decision on what action they should take is solely based on the local environment and a single global reward signal that relates to the global behavior of the compound population. From this point of view the problem of learning in development boils down essentially to a multi-agent reinforcement learning problem \cite{110,75}. Traditional approaches to multiagent learning have been to model the other agents’ behavior as a Markov decision process\cite{75} and then to compute the optimal response from the estimations. Here, we investigate more implicit ideas of local reward modulation and reduction of degrees of freedom.

Through learning we are now closing the large feedback loop \textbf{Local Structure} $\rightarrow$ \textbf{Global Structure} $\rightarrow$ \textbf{Global Behavior} $\rightarrow$ \textbf{Local Structure} discussed in Section 3.2.4. For this to work properly, we have to be sure that the global behavior is working as expected and thus, we used the most stable behavioral configuration. It turned out that inhibitory synapses and a simpler environment were better suited for the learning task because the behavior was more stable. With inhibitory synapses, the motor cells apply a forward force to the organism unless they are inhibited by axons

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{image.png}
\caption{Braitenberg vehicles with inhibitory synapses. Taken from \cite{21}. \textbf{a} is a \textit{Love}-vehicle because it approaches the good (in this case light) and \textbf{b} is a \textit{Fear}-vehicle because it avoids the good.}
\end{figure}
Figure 5.2: The learning rule for attracting the correct axon. The current strategy of the depicted motor cell is to release blue morphogen to attract blue axons. 

- A positive reward signal amplifies the current strategy by increasing the gain of morphogen production that is attracting the currently connected axon.
- A negative reward signal weakens the current strategy by decreasing the gain of the morphogen production that is attracting the currently connected axon. Continuous negative reward will yield a morphogen production of blue that is lower than red and give rise to the attraction of a red axon.

connecting from sensory cells that are active because they are sensing good underneath them (see Figure 5.1). In the Love task, the correct wiring pattern would be to connect in a parallel fashion: Left sensor cells connect to left motor cells and right sensor cells connect to right motor cells. This way, good sensed on the right side of the organism would inhibit the right motors and force the organism into a turn to the right and so increasing the input of good. On the other hand, the Fear task requires a crossing pattern for optimal behavior. A motor cell can choose one of the two strategies by either attracting axons from its ipsilateral or contralateral side.

## 5.2 A Local Structural Learning Rule

Figure 5.2 illustrates our learning rule. Structural learning is achieved by modulating the release of morphogens by the motor cells. Every motor cell carries two variables, each associated with one of the two strategies. These variables store the expected reward when executing their corresponding wiring strategy. We call these variables the confidences of the cell into its corresponding strategies. The confidence is updated according to the cur-
Learning in Development

\[ \Delta c_i = \eta (r - c_i) \cdot n_i, \]  

(5.1)

where \( c_i \) is the confidence in strategy \( i \), \( \eta \) is the learning rate and \( r \) is the reward received in the last time step and \( n_i \) is the number of connected synapses with type \( i \). We can see that in the equilibrium \( c_i \) is equal to \( r \) if the cell is connected to a cell of type \( i \). If the reward is negative, the confidence of the strategy associated with the currently connected axon is pulled below zero.

The rate of release of the morphogen associated with a strategy is proportional to its corresponding confidence. Hence, a cell having more confidence in particular strategy will express more morphogen associated with this strategy and thus is more likely to keep or receive a synapse of a cell from the corresponding sensory side: A positive confidence into axons of the blue type, will enable the production of the blue attracting morphogen, while a negative confidence will enable its consumption and so its removal from the environment such that axons of the blue type will no longer be attracted.

It can easily be seen that under idealized conditions, the correct wiring strategy is stable with this learning rule: A wiring that achieves the correct behavior will result in constant positive reward. The positive reward will reinforce the connectivity only for the currently (correctly) connected synapses and thus confirms this strategy. The confidences of the cells into this strategy will converge to the reward while the confidences into the opposing strategies will decay to \( \hat{c} \).

5.3 Caveats of Multiagent Learning

To test the learning rule we used the same description code as in Chapter 4. We removed the InterAxisReactor from the Motor cells and replaced the MotorCompetence by an AdaptingInhibitoryMotorCompetence. Its name contains Inhibitory because the activity from the incoming axons acts, as noted above, inhibitorily on the force this cell applies to the organism. This competence also contains the learning mechanism described above: it constantly evaluates its confidence in a given strategy and releases axon attracting morphogens depending on them.

The question was, whether a population of cells containing such a reactor would be able to find the correct wiring strategy without the InterAxisReactor that was responsible for the correct relative subdivision of the Motor population. It turned out that the performance was very bad, how-
5.3. Caveats of Multiagent Learning

Cooperative learning of independent agents turns out to be a very difficult problem due to several reasons:

1. The entire product space of all agent configuration needs to be searched if the agents’ actions are interdependent. I.e. if the global objective is not factorisable into the agents’ states. However, usually only a small fraction of all possible states fulfill the search objective.

2. The probability of reaching a good better state through random fluctuations decreases exponentially with number of agents. Because most configurations are non-functional, the objective landscape is very flat (Figure 5.3) and thus a gradient descent does not converge.

3. The behavior depends on a very large portion of agents being in the correct state. If only a few agents are in the wrong state and thus yield bad global behavior, also the correctly behaving agents are punished by the negative reward signal.

4. A single scalar reward signal that is applied to all agents simultaneously can not distinguish for a good credit assignment.

5. The reward depends on the global, longer timescale behavior and is delayed. It is difficult to determine which action actually led to the reward.

These problems can partially be resolved by the following measures:

1. Through self-amplification of behaviorally coherent global states the objective function could perhaps be ‘diagonalized’. These higher level modes in the organization[125] might have less degrees of freedom.
and can be varied as independent contributing factors of the objective function. A factorizable objective function can be optimized by searching each degree of freedom’s maximum independently. This mechanism is similar to the modularization employed by evolution (see Section 2.2.2).

2.-3. Reduce the variety (degrees of freedom) of the compound system through introduction of constraints on the states or by self-amplification of ‘more likely’ configurations. By the design and choice of ‘more likely’, however, we lose the objectiveness of the blind watchmaker.

4. Locally modulate the reward signal based on the agent’s estimated participation in the global action that caused the reward.

5. Incorporate the changes in the timescale of the behavior and reward by using a very small learning rate. This will result in slow learning.

5.4 Local Reward Modulation

To optimize cooperative behavior in systems of many individual agents but only a single reward signal, different approaches have been made: Chang et al.[25] suggest to use Kalman-filters to estimate local rewards based on the global reward signal, Guestrin et al.[53] suggest to use coordinated actions and communications between the agents to optimize their policies, and finally Bagnell et al.[11] point out that solving multiagent reinforcement learning tasks is almost infeasible when not allowing any local rewards.

As pointed out in bullet 4 above, one way to resolve the credit assignment problem is to locally modulate the reward before using it to learn. In this scheme an agent should locally scale the reward by an estimation of its participation to the action the lead to the global reward. Often it is impossible for an agent to have knowledge about its exact influence on the global behavior. The extent to which multiagent learning without local reward is possible strongly depends on the kind of task and actions involved in the problem. Some problems may allow single agents to cleverly extract quite well a good estimation of their local participation in the global reward. In our example of the learning motor cells we can use the following estimation:

If a motor cell is using the opposite strategy than its neighboring cells, its effect on the global behavior is likely to be weakened because the neighboring cells are acting against it. However, if a cell is using the same strat-
5.4. Local Reward Modulation

ey as its neighboring cells, its effect is not canceled out and thus it has reason to believe it has a larger influence on the global behavior. We can therefore use the similarity of our confidence to the neighbors confidences as scaling factor for the reward. This was assumed to work particularly well because the cooperative local confidence could easily be read off in the local concentration of the attracting morphogen because the motor cells are releasing the attracting morphogens proportional to their confidence.

5.4.1 Simulation

We tested this idea in a simple MATLAB simulation where a vector \( \mathbf{x} \) of 100 linearly arranged cells had to choose -1 or 1 to match a given template vector of the form\(^1\)

\[
\mathbf{t} = \{ t_j \}_j = \begin{cases} -1 & \text{if } i \leq 50 \\ 1 & \text{if } i > 50 \end{cases}
\] (5.2)

The system had to match its configuration vector \( \mathbf{x} \) to the template \( \mathbf{t} \). Each component of \( \mathbf{x} \) is one cell that follows a strategy (be -1 or be +1) and has confidences \( c_{-1} \) and \( c_{+1} \) in these strategies. At every step, each cell reconsider its current configuration. The new state is drawn probabilistically according to the Gibbs probability distribution of its current confidences \( p_i = \exp(-c_i/T)/Z \), where \( T \) was set to 0.1 and \( Z \) is the partition function normalizing the distribution.

The reward is computed as the inner product of the state and the template vector:

\[
r = \langle \mathbf{x}, \mathbf{t} \rangle / 100.
\] (5.3)

The reward is therefore a scalar between -1 and +1 signifying the similarity between the actual state and the template.

The cells update their confidences for the states -1 and +1 according to equation (5.1). This update requires a reward signal. \( n_i \) is characteristic for the current strategy \( n_{-1} = 1, n_{+1} = 0 \) if \( x_i = -1 \) and vice versa in the case \( x_i = +1 \).

To test the local reward modulation we introduced local similarity. For every cell, the local similarity is computed by a convolution of the state vector with a normalized Gaussian kernel. Before applying update (5.1) to their confidences, the cells scale their reward \( r \) proportional to their entry in the similarity vector thus locally modulating their reward.

\(^1\)Any template with local similarity could be learned this way. Here, we chose this simple one because it closely resembles the correct configuration of the motor cells in our organism.
Figure 5.4 shows the convergence behavior of the simulation depending on different kernel sizes. In a population of 100 linearly arranged cells, a local view of as few as 10 cells across, with the bulk of weight on the closest 10 neighbors (5 on each side) is enough to maximize the convergence performance at around 100 steps. The speed of convergence is measured in number of steps until the population reaches a reward higher than 0.85 for the first time. We can see that a local reward modulation can be beneficiary if we use an appropriate estimation. If the kernel is too large, the performance decreases again. However, the graph shows a clear tolerance to the exact size of the kernel and hence to the extent of the local neighborhood being considered for the local estimation.

### 5.4.2 Organism

From the tolerance with respect to the size of local kernel seen in the simulation presented in the last subsection, we expected a well-tempered convergence behavior when plugging the learning rule into the motor cells of our developing organism. Unfortunately, it turned out that the behavior of a partially developed organism that hasn’t learned the entire correct
connectivity yet was so bad that the reward signal generated from it was entirely unrelated to the state of its development and the quality of its intermediate wiring. The system was subject to caveat 2 from Section 5.3: The search space is too big and flat. No organism developed properly.

5.5 Reduction of Degrees of Freedom

Following the resolution of caveat 2 from Section 5.3 we reduced the dimensionality of the search space by adding an AxisReactor to the Motor cells. This is a similar mechanism as proposed by Willshaw and von der Malsburg\cite{125, 118} to drive the system to higher modes through self-amplifying fluctuations. By contrast to the description code in Chapter 4 where we used an InterAxisReactor, the AxisReactor still leaves open the orientation of the principle axis of the motor population. It can be either parallel or anti-parallel. The inclusion of the AxisReactor reduces the variety of configuration space from $2^N$, where $N$ is the number of motor cells to $2^1$ (parallel, or anti-parallel) which simplifies the problem significantly while still leaving a fact about the world to learn. Both of the two states are clean global configurations of cells that give rise to globally coherent behavior. From the coherent behavior we are now able to generate well-mannered reward signals thus resolving the problem of Section 5.4.2. Because now in a coherent global configuration, every cell has a proper influence on the global behavior and therefore we do not anymore need to include the local reward modulation.

Figure 5.5 shows two developed example organisms and the performance of the development. The two displayed developed organisms are wired up correctly according to their task and behave as expected. The graph of subfigure c shows the performance of the learning development. The behavior is measured as the fraction of the path length the organism spent on the good in 500 simulation steps. The developed organisms show a significant bias in their behavior towards their trained task: Love-organisms spend on around 60% of their time on the good while Fear-organisms only 20%. The Love-organisms’ behavior is much more spread out in performance because of instabilities in the line following and the bias towards circling.

In general, the learning works. However, there are also a few outliers in the Fear task. The two most extreme outliers are two organisms that have developed into Love-organism despite the punishing reward signal from its behavior. The closer four outliers have correctly developed into Fear-organisms but are slightly unbalanced and thus their behavior does
Figure 5.5: Performance of learning.  

**a** A developed organism for the task *Love*. The environment is shown right of the organism. The two green bars are good, scattered in the environment. White dots depict the track of the organism. In the *Love* task, the organism has to dwell on the good.  

**b** A developed organism for the task *Fear*. In this task, the organism has to avoid the good.  

**c** A boxplot of the behavior of the developed organisms. N=69 in both tasks. The box displays the median, upper and lower quartile of the fraction of path length spent on the green good. The whiskers show the extent of the data up to 1.5 of the interquartile length and the outliers beyond the whiskers are displayed as crosses.
not qualify as *Fear* behavior.

## 5.6 Conclusion

Learning structure turns out to be a difficult problem: There are so many possible structural configurations – how can local processes find the right one that leads to the desired global behavior? Essentially, learning in developing structure can be looked at as a multiagent reinforcement learning problem where the individual cells are autonomous agents, and the global behavior of the organism gives rise to the reward signal. We saw a number of caveats that can be encountered when trying to learn structure. Like in the development scheme itself, the idea should be to find a mapping from global to local: Understanding how the local structure influences the global behavior can help to extract a correction signal for the local structure based on a global reward signal.

Because the reward is a single global signal that is applied equally to all learning cells it is difficult to assign credit to a particular strategy of a single cell. We introduced the notion of *local reward modulation* by which an agent locally modulates the global reward by a factor that estimates the agents participation in the global reward. We saw that, with a well-behaving configuration-reward mapping, a scheme like this can be well converging and robust against estimation errors.

However, it is crucial that the reward signal generated by the global behavior is a response graded by the quality of the structure. Because there are so many configurations of agents possible a large portion of the configuration space must yield graded reward signals that can be used to perform a gradient descent on the reward landscape. If this is not the case, as it was in our organism, convergence to a well behaving organism is very unlikely. In a case like this, we need to help to stabilize the behavior to ensure a well-mannered reward signal. We introduced a self-organizing process that helped us in two ways: Through the partitioning of the motor population (1) it guaranteed a coherent structure that gave rise to stable behavior and thus a good reward signal. And (2), it reduced the degrees of freedom of the learning problem to 1, parallel or anti-parallel which greatly simplified the problem.

We saw that the inclusion of the self-organizing process that reduced the degrees of freedom and stabilized the behavior lead to an acceptable performance of the development. However, it is not clear, how the solution of general problem could be learned in a developmental scheme like this. It seems that the extent to which the structure needs to be prespecified in
the description code and how the degrees of freedom should be reduced through self-organization or other constraints needs to be evaluated on a problem by problem basis.
Chapter 6

Networks of Processes

6.1 Introduction

On the verge breaking Moore’s law, we are now close to reaching the upper bound of computational power on a single processor. As a consequence, engineers are aiming at parallelizing computational problems. Tasks that can be segmented into independent computational chunks can easily be distributed across different CPUs with low-bandwidth interconnectivity\[66\]. Other parallel algorithms exploit algebraic properties and symmetries of mathematical systems to distribute them equally on a matrix of processors. However, for most computational problems it is non-trivial to derive an efficient distributed algorithm.

Nervous systems lie at the extreme of distributed computing. The computational capabilities of a single cell is minimal and the bandwidth of its neurites is very low compared to that of electronic circuits. The computational power relies on the massive dimensionality of the system and the complexity of its wiring. Their computation is inherently different to engineered computation by utilizing huge amounts of simplistic computational modules that are asynchronously working to exploit cooperative effects.

**Graphical models** are based on similar principles like nervous systems: their computation is achieved by a graph of communication processors that perform very simple computations\[62, 68\]. Prominent members of graphical models are factor graphs and the class of belief propagation algorithms, Bayesian networks and random Markov fields\[61\]. Although most graphical algorithms are run synchronously, in the sense that waves of messages sweep the graph, most of them can also be run asynchronously by performing incremental computations on the nodes. Graphical models are very well suited for distributing algorithms across networks because they don’t need
synchronous updates, but are event driven. Also, they cooperatively converge to the solution of a problem while keeping the load at any particular node minimal.

In the previous chapters we have looked at the development of an artificial multicellular system. In this chapter we will abstract some of the principles derived earlier and apply them to the problem of distributed computing and sensing. Roughly, we implement the following transition:

- Physical substrate → Computer network
- Cells → Processes
- Chemical diffusion → Message passing

As before, the organization of the processes, and the computation that arises from this organization is guided only by local signaling. There will be no global observer. Processes live on the nodes of the network and are executed by the scheduler on the node. None of the nodes know the global network topology a priori, the processes can send out messages that are transported through the link structure composed of the network nodes and the connections between the nodes. Because chemical diffusion is very closely related to random walk of particles, the diffusion of information can be easily mimicked by network messages traveling randomly across nodes.

In section 6.3 we will describe a well suited, distributed routing algorithm for routing messages in networks of unknown and changing topology. We will need this as the basis for later stages. In section 6.6 we develop a self-construction scheme for embedding graphical models inside a network. This embedding attempts to optimize processor load and communication cost between communicating nodes. And in sections 6.6 and 6.7 we will explore a solution for distributed sensing solving the following problem: We are given a network of unknown and changing topology. The nodes in the network carry sensors that sense properties of the real world. We present a scheme for processes to dynamically divide, distribute and interconnect themselves to build a causal model of the world that can predict the values of the sensors.

### 6.2 The Processes and Messages

The processes that live on the network are structured as illustrated in Figure 6.1a. A process consists of a core and modules that can be dynamically added to the core. The core of the process maintains the communication
6.3 Dynamic Routing

In order to optimally transport information from one process to another, located at some other node in the network each node of the networks needs to carry routing information based on which it can pass messages on to its neighboring nodes. In practice, this means, that each node keeps a bias for each destination associated with every channel that connects it to another node of the network. The biases encode the value of sending a message.
to a specific destination along the associated link. In an optimally routing network, every node will send its messages along the link that constitutes the shortest path to the destination.

6.3.1 Previous Work

Bellman-Ford routing

Bellman-Ford routing[16] (BF-routing) is the most commonly used routing algorithm. It is the basis of almost all today’s used routing schemes, most notably the routing information protocol (RIP), the most popular IP-routing algorithm used on routers in the Internet. In classical BF-routing, a global observer computes the shortest path between any two nodes and stores this information in a lookup table at each node. A network package passing a node is sent on to the next node based on this information and thus chooses a globally optimal route.

Q-Routing

Similar to the way the reinforcement learning variant Q-learning is the local, adapting version of dynamic programming, there is a local and adapting version of BF-routing called Q-routing. It was first introduced by Littman and Boyan[74]. Like in BF-Routing, in Q-Routing, every node x of the network carries a table of Q-values that stores the values for sending off a message with destination d to neighboring node y, Q_x(y, d). This time, however, these values encode a function of the estimated time (instead of the globally computed time) for a message to reach its destination d when passed on to node y and then following the current optimal routing strategy. When a message is sent on to node y, the node x receives back the update value Q_y(ẑ, d) = \min_{z \in N(y)} Q_y(z, d) from node y; here N(y) denotes the set of neighbors of node y. In terms of reinforcement learning, Q_y(ẑ, d) plays the role of the expected future rewards. Node x will then revise its Q value based on the following update equation:

\[ \Delta Q_x(y, d) = \eta(Q_y(ẑ, d) + q_y - Q_x(y, d)), \]  

(6.1)

where q_y is the delay the message encounters when traveling from node x to node y and \( \eta \) is the learning rate. In a converged network, where the \( \Delta Qs \) vanish, each node carries Q values that fulfill

\[ Q_x(y, d) = Q_y(ẑ, d) + q_y. \]  

(6.2)
In other words, every Q value carries the remaining time a message has to travel to reach its destination when following the current optimal strategy of being sent on to the node y with minimal latency \( \arg \min_{y \in N(x)} Q_x(y,d) \).

In order to not get stuck in a local optimum, instead of using the greedy strategy that selects the Q with minimal latency, a good routing policy should select the next nodes by using a soft min weighted by the actual Q values. Exploration and update of this kind is called forward exploration.

**Dual Reinforcement Q-Routing**

Kumar and Miikkulainen\[70\] noticed that Q-routing could be made to converge faster, if at every hop, not only the sending node but also the receiving node would update its values. The message traveling from source s to destination d can tell a node y the estimated remaining time to travel to the source s: \( Q_y(x,s) \), and node y can revise its Q values according to:

\[
\Delta Q_y(x,s) = \eta(Q_x(\hat{z},s) + q_s - Q_y(x,s)).
\]

This update adds exploration in the reverse direction of the traveling message and is thus called backward exploration. The nice thing about backward exploration is that the update value \( Q_x(\hat{z},s) \) can be carried by the traveling message and so the only overhead resulting from backward exploration is a slight increase in the message size. No additional messages have to be sent. Kumar and Miikkulainen have termed the dynamic routing with forward and backward exploration Dual Reinforcement Q-Routing (DRQ-Routing) with reference to Dual Reinforcement Learning, which was first used in satellite communications\[48\].

**6.3.2 Age Based Q-Routing**

Independently of the work presented above we developed a routing algorithm that fits nicely into the framework of Q-Routing: age based Q-Routing (Figure 6.2). Our algorithm is based on the fact that even if randomly diffusing, messages are more likely to arrive at a node through a channel that connects to a node that is closer to its source s. To send a message to a specific process d, it is therefore a good strategy to send it off in the direction where most messages with source d came from. Two-way links are self-enforcing: If all messages from A to B traverse on the same path, they bias the traversed nodes to send off messages traveling from B to A along the same links in the opposite direction. This again will strengthen the path in the original direction A to B.
Networks of Processes

Figure 6.2: Message based updating of routing information. Left: A message $P_{ab}$ traveling from $a$ to $b$ arrived at the center node. Each node carries an associated Q-value for destination $b$ for every connection. Middle: Based on a weighted soft-max decision, the message is passed on to the next node. In this case the node with Q-value 20 won the soft-max. Right: The receiving node now updates its Q-value for destination $a$ based on the traveling time of message $P_{ab}$. Notice that not the Q-value on the sending node that took the action is updated, but the receiving node.

The influence of a message on the Q-value of the nodes decays with time, such that messages traveling on a shorter path have a stronger influence. A simple Q-value update rule contains a decay and an influence from the message that is passing. If a message $M_{s \rightarrow d}$ is arriving through a channel that is connecting node $x$ to the current node $y$, the Q-value $Q_y(x, s)$ is updated according to:

$$
\Delta Q_y(x, s) = \eta \cdot (\exp(-t_M\alpha) - Q_y(x, s)),
$$

where $Q_y(x, s)$ is the Q-value associated with the message source $s$. $\eta$ is a small learning constant, $t_M$ is the age of the message and $\alpha$ is an aging factor. Notice, that (6.4) is essentially the backward exploring update (6.3), with $Q_y(\hat{z}, d) + q_y$, which is an estimate of the traveling time replaced by $\exp(-t_M\alpha)$ which is a function of the actual traveling time. So in this algorithm, instead of relying on estimates of previous nodes we can bias every node based on facts that are known by the message. Now the values are inversely proportional to the latency – meaning, higher values are better for routing.

However, such an update rule has the problem, that suboptimal routes can be stable equilibria of the system. If a suboptimal route is active and hence self-enforcing, a message traveling on a more optimal path is very rare because the bulk of the traffic is going through the self-enforcing route as chosen by the soft-max in each node. A single message traveling on
a better path will not be able to re-bias a node of the original path to send future messages in its direction because the update rule is incremental and small compared to the original bias. We need an update rule that immediately re-biases a node if a better route presents itself. Instead of slowly converging against the equilibrium bias with $\Delta Q_y(x, d) = 0$, we can allow better messages to immediately re-bias the node to the equilibrium:

$$Q_y(x, s) \leftarrow \max(Q_y(x, s), \exp(-t_M \alpha)).$$

(6.5)

To find the optimal route, the algorithm will only update $Q_y(x, s)$ if the new value is greater than the old one. The max clause will ensure the memorizing of value of the message with the shortest latency. If, however, the topology of the network changes through time, the optimal route could change and the currently stored Q-value might no longer be valid. To prevent this from happening, the Q-values must decay with usage: every time a message passes through a node, all Q-values of the node are decayed by a decay factor. As long as messages travel on the optimal route, they constantly refresh the Q-value against the decay. As a consequence, the Q-values will converge to the value of the path they are currently part of. On the other hand, if the optimal route changes, for example due to changes in the network topology by removing an edge or adding an edge, the Q-values will decay or rise to their new maximum value and correctly reflect the optimal routing policy.

### 6.3.3 Results

We measured the performance of our routing algorithm on a simple test network (Figure 6.3). There are four processes in the network: processes 1 through 4, where process 1 is communicating to process 4 and vice versa, and 2 is communicating to 3 and vice versa. Edges have a limited transport capacity such that if more messages travel on the edge in unit time (i.e. the load is higher), the messages have longer delays until they are delivered. A bottleneck joins the left and the right subnetwork. We compared the message delivery times for a network with one connection and a network with two connections between left and right. A simple shortest path routing algorithm that finds the shortest path based on network topology would be unable to distribute the network load onto two network connections comprising paths of suboptimal lengths because it is insensitive to actual delivery times. However, age based Q-Routing will optimize the compound delivery time through correct Q-value assignment and its consequent routing.
Networks of Processes

Figure 6.3: Two simple test networks. Filled circles are network nodes, lines are connections between network nodes, where communication occurs. The numbers describe the locations of processes 1 through 4. **Left:** A network with only a single connection from left to right. **Right:** A network with two connections from left to right. A shortest-path routing algorithm would be unable to distribute the network load on both connections.

Figure 6.4 shows the routes for the two pairs of communicating processes (1-4) and (2-3). The displayed routes are computed taking the direction of maximal Q-value at each node (greedy routes). However, age based Q-Routing soft-maxes the Q-values and thus stochastically finds its routes. These routes fluctuate around the greedy routes. In the case of a single connection between the left and the right subnetwork, the routing algorithm finds the shortest path route. In a rectangular grid, the shortest path is ambiguous: many routes have the same length. From those routes, the algorithm chooses non-overlapping routes for the communications of the two pairs of processes. Because the optimal routes are ambiguous, there is a large fluctuation in the choice of the route among the shortest paths. If the algorithm accidentally finds a different shortest path route it enforces this route in the same way as the original route.

If a second connection is added (Figure 6.4b, one of the two pairs of communicating processes takes the penalty of detouring through a longer route. It does so because using the overlapping shorter route increases the load on the connecting edge and thus significantly delays the delivery time of both communicating pairs. This delay penalizes both communicating pairs and thus reduces the Q-values on the nodes such that a small detour still turns out to be a better solution. Figure 6.5 shows a graph of the total average latencies in the two networks. The performance of the ‘1 connector’ routes is equivalent to the shortest path route. The latency of the ‘2 connectors’ routes is significantly lower and thus better in performance than the ‘1 connector’ routes because the network traffic is distributed among the two
connections and thus the congestion on a single node is minimized.

Each communicating pair is selfishly optimizing its own delivery time. In the given network, this way, it optimizes the compound delivery time. There are cases where selfish routing can lead to suboptimal global behavior [99, 20]. However, it has been shown that in networks where the edge latency is a linear function of its congestion, selfish routing leads to a total latency of maximally $4/3$ over the optimal minimum latency [99]. Thus, if a local algorithm is used for finding the optimal route for any user of the network, the global latency of communication processes is near optimal.

### 6.4 Process Migration

The module that implements the mobility of the processes is the migration module. Based on locally accessible information it decides to migrate or not the process to a neighboring node in the network and thus refining its position with respect to information sources in the network and the overall structure of communication of the distributed system.

The network routing algorithm presented above provides the basis for a topology: Each node $x$ carries a value that is relating its location to any
Figure 6.5: The average transmission delays in the two networks. In both networks, the algorithm needs an initial bootstrapping to find an optimal route in the given network topology (bump between $0.2$ and $0.3 \times 10^5$ time units). After the bootstrapping, the routes equilibrate around their optimal solution. The routes in the ‘2 connector’ network have significantly lower transmission delay than the ones in the ‘1 connector’ network because the algorithm can distribute the traffic along the two connections between the left and the right subnetworks.
particular process $i$ that is distributed on the network:

$$V_i(x) = \max_{y \in N(x)} Q_y(x, i) \quad (6.6)$$

$$\approx \exp(-t_i(x) \alpha). \quad (6.7)$$

The second step follows from (6.5), $t_i(x)$ is the minimal latency of a message traveling from node $x$ to process $i$. $V_i(x)$ is thus related to the distance between $x$ and process $i$. Hence, these values provide information for a distance metric on the network that is related to current positions of all the processes. Such a metric gives rise to a topology that can be used to guide optimal placement of processes with respect to one another. In particular, the topology is accessible locally and thus processes can use it for migration.

In order to optimize co-localization to other processes, the migration module of process $i$ can carry a migrational objective $O_i(x)$ which is a function of the values $V_x(j)$ of the node $x$. The migrational process aims at maximizing this objective function. It does so by sending out a migration query to a randomly chosen neighboring node $y$. The neighboring node evaluates the processes objective function at its position and returns the query. The migrating process can then compare the neighboring objectives to its own and decide whether it chooses to do a migrational step or not. This is implemented in a Monte-Carlo way: If $O_i(y) > O_i(x)$, a migrational step will be taken with certainty (we always move to better places). If $O_i(y) < O_i(x)$, then the step will be taken with probability $p = \exp(-(O_i(x) - O_i(y))/T)$. Just as in the cell migration of Chapter 4, this will reduce the likelihood of converging to a local minimum.

If a process aims to be close to a connected process $k$, the objective function should include a term for the value $V_x(k)$. Most of the time, however, processes want all the communication latencies to their connected processes to be minimal but they should all be similar in length. To avoid gathering of several processes on a single node, there should be a term penalizing the number of processes in the objective function. This gives a function of the form:

$$O_i(x) = \sum_{j \in \mathcal{C}} \langle l_j \rangle - \sum_{j \in \mathcal{C}} \sqrt{\langle (\Delta l_j)^2 \rangle} - \pi n_x, \quad (6.8)$$

where $l$ is the vector of latencies $\{\log V_j(x)\}_{j \in \mathcal{C}}$ of the connected processes $\mathcal{C}$ and $\pi$ is the penalty factor and $n_x$ the number of processes on node $x$. $\langle l_j \rangle$ and $\sqrt{\langle (\Delta l_j)^2 \rangle}$ are the mean and standard deviation of the $j$ths component of $l$. According to (6.5), $\log V_i(x)$ is proportional to $-t_{M_i}$ which is
Figure 6.6: Process migration, from top left to bottom right. Processes 1 and 2 are fixed and process 3 is allowed to migrate. The discussed objective function lets process 3 migrate into the middle of the two connected processes 1 and 2 thus minimizing communication latency.
the negative age of the message from the connected process \( i \). If \( O_i(x) \) is maximal, the message ages are thus minimal and the scattering of the ages is also minimal. I.e. A process attempts to be close, but equally distant to all its connected processes (Figure 6.6).

However, the value imposed topology is very much dependent on the current routes the messages choose for communication. Close to such routes, the topology is approximately correct, but further away it doesn’t need to be anymore. Figure 6.7a shows a situation with distorted topology. Because process 1 is swallowing all messages coming from process 2, behind process 1 the values for process 2 are 0. Figure 6.7b shows a simplified situation where this leads to a problem. To illustrate the case, we use values that are linear in distance and no penalty term in the objective function. Process 3 would like to center itself between process 1 and process 2. The objective function at its current location is

\[
O(x) = \frac{\text{latencies}}{4} + \frac{2}{3} - \frac{\text{scattering}}{1} = 5.
\]

The objective function on the middle node would be

\[
O(x) = \frac{\text{latencies}}{3} + \frac{3}{3} - \frac{\text{scattering}}{0} = 6,
\]

which is better than 5. However, because the topology is distorted due to the swallowing effect, the middle node currently has an objective function of

\[
O(x) = \frac{\text{latencies}}{0} + \frac{3}{3} - \frac{\text{scattering}}{1.5} = 1.5,
\]

which makes it very unattractive to migrate to. Fortunately, in networks with more than 1 dimensional topologies such artifacts are not as strong because there is always diffusion around a swallowing node along the other dimensions. Similar artifacts are seen around optimal routes. Because they are self-enforcing, hardly a message ever tries a different route and thus the nodes away from the routes don’t get correct value information. The temperature parameter regulates the randomness in the choice of the message routing. An infinite temperature would yield a perfect random walk of messages which is equivalent to homogeneous diffusion. This would be beneficiary for the topology because on average, all nodes receive a correct unbiased estimate of their distance to the processes. However, a random walk routing strategy would not be very efficient for communication between processes. It is therefore the right balance in the trade-off between routing and migrating that provides optimal performance.
Figure 6.7: Problems of migration on a 1-dimensional network. The number above the nodes denote processes. The curved lines symbolize their communication. a exemplified values $V_2(x)$ of network nodes $x$ relating to process 2 are written underneath the nodes. The node that is hosting process 2 has highest value 6. Every step away decreases the values. Because no messages from 2 to 1 travel beyond the node of process 1, the values behind the node of process 1 are 0. b Centering process 3 between process 1 and process 2: Process 1 is communicating to 3 and process 2 is communicating to 3. The values $V_1(x)$ are in the top row and $V_2(x)$ in the bottom row underneath the network nodes. Due to the distorted topology as seen in a, process 3 is unable to center itself.

### 6.5 Graph Construction

As discussed in the introduction to this chapter, graphical models are well suited for distribution across networks. In this section we investigate the automatic embedding and maintenance of a graph of connected processes in networks. Figure 6.8 shows an example graph that we want to embed into a network. Each node of the graph is represented by a process. The edges of the graph are communication channels between the processes. Similar to biological development, we want the graph to grow from a single process and optimally arrange itself according to the network topology. We won’t allow a global supervisor orchestrating this construction process.

To achieve this, we have to add two additional capabilities to the processes:

1. to contain an internal description of the entire graph
2. to replicate into two processes.

These two capabilities are implemented in the **Graph Module**.

Figure 6.9 shows the growth of a graph from one single node of the graph is placed into the network. According to its graph encoding the first
process will try to establish a connection to its neighboring graph nodes. If this fails, it will divide and the newly born process will become its neighboring graph node. The newly born processes will behave in exactly the same manner and thus, incrementally, the graph will unfold. Like in the development of the multicellular organism of chapter 4, processes are responsible for themselves and their immediate environment. By following local rules of construction, they build the whole system. The system is self-sensing and each component understands what actions it needs to take to become locally complete: the system is thus self-constructing and self-maintaining.

Figure 6.10 shows the behavior of the graph if the network topology is altered. The connection between process 3 and processes 2 and 5 is temporarily interrupted (b,c). The processes find new routes through the network (d). Migration then occurs along the routes of communication until the graph is optimally arranged on neighboring nodes (e). However, when the network is cut into two unconnected parts the graph is separated into two subgraphs that can never reconnect. The processes responsible for the cut connections will divide again and recreate the lost processes thus rebuilding the entire graph. This happens in both partitions of the network and hence, two separated copies of the graph are rebuilt (f through h).

For the implementation of the graphical model, the processes should of course also carry a functional module that is concerned about the content of the messages and the local computation the process has to perform. In this thesis we are not concerned about the particular implementation of the graphical model, we merely demonstrate how to set up a self-maintaining structure that is capable of such computation. Explicit graph construction, as demonstrated here, could very well be used in distributed systems where the circuitry is known a priori, but not the network it will be embedded in. More complex distributed systems could be of a hybrid form where some
Figure 6.9: The growth of the graph of figure 6.8 in a network of a 2D grid. Time evolves from top right to bottom left. In all plots we see the connectivity of process 3.
Figure 6.10: Optimal arrangement of the graph. If we cut the network, the processes of the graph will migrate and arrange themselves to optimize communication. If the network is cut and no communication between some of the nodes is possible anymore, the graphs will rebuild their missing parts due to their local rules of construction: Two equivalent graphs are now constructed.
of the connectivity needs to be learned (as we will see in the next section) and some of the connectivity is given by the designer.

### 6.6 Connectivity from Observation

In the last sections we have introduced a framework of self-constructing and self-arranging graphs of mobile processes. As we have already seen earlier in chapter 5 self-sensitive structures can dynamically adapt their structure based on information retrieved from their already set up structure and their immediate environment: they can construct themselves to fit the world they live in.

In this section we will use this capability to design a system that is able to build a simple model of the world it lives in. We attempt to build a system as illustrated in Figure 6.11. We consider the world to be a deterministic system with complex but causal dynamics. The abstract world where our model lives in is limited to the network the processes are embedded in. The network has access to a limited number of observables from the world. Based on these observables, we attempt to build a graph of connected processes that reflects the causal connections of the world. From such a constructed model it is then possible to estimate the state of obscured or hidden variables. Because we assume the mechanics of world to be constant, a constructed model that hasn’t contradicted the observed data so far is likely to be a good model for the future. In any case, objectively, it is the best we can do from what we have observed. Hence, we can use the graphical model to infer states of the future.

We restrict our world to be a network of probabilistic relations (Figure 6.12). Networks of probabilistic relations can be represented as directed acyclic graphs (DAGs). The observables are Boolean variables. In causal networks, the directed edges (arcs) specify causal relations between the observables. An arc connecting observable 1 to observable 2 means, 1 is causing 2. In this case, whenever observable 1 holds true, 2 must hold true too. Additionally, an arc can carry a conditional probability of cause, in which case the the network is not strictly causal anymore, but rather Bayesian. We allow hidden variables to be able to cause our observables too, hence, every observable could be true without an observable cause. The table in Figure 6.12b shows the set of possible states of the world caused by the causal relations of the graph in subfigure a.

Because observable 9 is causally unconnected to the rest of the world, its value does not interfere with the rest and thus for every state of the world without 9, there is a possible state with 9 true and with 9 false. Thus
6.6. Connectivity from Observation

Figure 6.11: Building a model of the world. In the top part of the figure is an example of the complex machinery of the world (Sculpture “Heureka”, 1964 by the Swiss Artist Jean Tinguely). The lower part of the figure depicts the abstract world in which we want to build a model of reality. Some states of the world are observable, we call them observables (red). Observables can be measured and transported into the modeling world (red arrows). Just from observing the observables we attempt now to build a model (blue) of the world above. From this model we want to be able to predict future events.
Figure 6.12: a A directed acyclic graph (DAG) of causal relations. The nodes are observables, the arcs are causal relations. For example: observable 2 is caused by observable 1. b Table of possible states of the world corresponding to the graph of a. 1 means true, 0 means false, * means either value is possible. There are 22 possible states (11·2).

This world can have 22 different states. 22 states is significantly less than the theoretically possible $2^9 = 512$ states if all observables are unrelated. This reduction of degrees of freedom corresponds to the constraints that are given by the laws of the world.

The question is now: Just by observing these possible states of the world, can we infer these constraints, the causal relationships that connects the observables? Mathematically stated, if $D$ is the set of data observed in the world (the set of possible states), we want to find a graph $G$ that best matches the data $D$. There are several algorithms on how to do that[100]. It can be done by maximizing the likelihood $L(G|D) = P(D|G)$ that the graph $G$ actually produces the data $D$. Or by Bayesian inference [43, 88, 89] where $P(G|D)$ is explicitly computed from the measured statistics $P(D)$, the known conditional probability distribution $P(D|G)$, and an assumed prior probability distribution $P(G)$. Through the prior probability distribution a penalty can be added into consideration for the complexity (and thus the prior probability) of the graph $G$. Inferring conditional probabilities from sample will always also include indirect causes. Pearl[88] and others has developed an algorithm (the IC-algorithm) that aims at extracting only direct causes from sampled data. It uses an implementation of Occam’s razor that states that if several models can account for the data, then “it is reasonable to rule out any model for which we find a simpler,
6.6. Connectivity from Observation

less expressive model, equally consistent with the data.” However, the IC-algorithm is global non-incremental, and thus not suitable for the purposes of implementation in networks of processes.

In the following, we present a learning rule that attempts to extracts the conditional probabilities between any two observables implemented in the networks of processes introduced earlier in this chapter. If we assume the world to be strictly causal and not probabilistic, the conditional probabilities can be used to infer a likely causal relation by simply assuming a causal relationship of the conditional probability exceeds the arbitrary threshold value of 0.8.

6.6.1 A Causal Learning Rule

We will present now an incremental learning rule for the probability of cause between observables $x_i$ and $x_j$ by observation of data sampled from the world. The probability of cause $P_{i\rightarrow j}$ is the conditional probability $P(x_j|x_i)$, hence:

$$P_{i\rightarrow j} = P(x_j|x_i) = \frac{P(x_i, x_j)}{P(x_i)} \approx \frac{f_{ij}}{f_i},$$

(6.9)

where $f_{ij}$ is the frequency of the joint occurrence of $x_i$ and $x_j$ and $f_i$ is the frequency of occurrence of $x_i$ by itself. The last step is the empiric frequency approximation to probability. For every event tuple $(x_i, x_j)$, $f_{ij}$ and $f_i$ are updated accordingly in the following way:

$$f_{ij} \rightarrow f_{ij} + x_i \cdot x_j$$

(6.10)

$$f_i \rightarrow f_i + x_i,$$

(6.11)

i.e. the frequencies are increased if their corresponding event happened. From this we can derive an update for the probability of cause from $P_{i\rightarrow j}^{\text{pre}}$
to $P_{i \rightarrow j}$. The goal is to specify $P_{i \rightarrow j}$ in terms of $P_{i \rightarrow j}^{\text{pre}}$.

$$P_{i \rightarrow j} = \frac{\bar{f}_{ij} + x_i \cdot x_j}{\bar{f}_i + x_i}$$  \hspace{1cm} (6.12)

$$= \frac{\bar{f}_{ij}}{\bar{f}_i} + \frac{x_i \cdot x_j}{\bar{f}_i + x_i}$$  \hspace{1cm} (6.13)

$$= \frac{\bar{f}_{ij}}{\bar{f}_i} \left(1 + \frac{x_i}{\bar{f}_i}\right) + \frac{x_i \cdot x_j}{\bar{f}_i + x_i} \left(1 + \frac{x_i}{\bar{f}_i}\right)$$  \hspace{1cm} (6.14)

$$= \frac{\bar{f}_{ij}}{\bar{f}_i} \cdot \left(1 - \frac{x_i}{\bar{f}_i}\right) + \frac{x_i \cdot x_j}{\bar{f}_i + x_i} \cdot \left(1 - \frac{x_i}{\bar{f}_i}\right)$$  \hspace{1cm} (6.15)

$$= \frac{\bar{f}_{ij}}{\bar{f}_i} - \frac{x_i \bar{f}_{ij}}{\bar{f}_i + \bar{f}_i} + \frac{x_i \cdot x_j}{\bar{f}_i + x_i}$$  \hspace{1cm} (6.16)

$$= \frac{\bar{f}_{ij}}{\bar{f}_i} - \frac{x_i \bar{f}_{ij}}{\bar{f}_i} + \frac{x_i \cdot x_j}{\bar{f}_i + x_i}$$  \hspace{1cm} (6.17)

$$= P_{i \rightarrow j}^{\text{pre}} + \frac{x_i}{\bar{f}_i} \left(x_j - P_{i \rightarrow j}^{\text{pre}}\right).$$  \hspace{1cm} (6.18)

The step between (6.14) and (6.15) is done by linearizing $1/(1+h) \approx 1-h$ for small $h = x_i/\bar{f}_i$ because $\bar{f}_i \gg x_i$. The step between (6.16) and (6.17) is done by disregarding the last term which is in the order $O(h^2)$. The learning update is thus given by

$$\Delta P_{i \rightarrow j} = \frac{x_i}{\bar{f}_i} \cdot \left(x_j - P_{i \rightarrow j}^{\text{pre}}\right).$$  \hspace{1cm} (6.19)

It depends on $x_i, x_j, \bar{f}_i$ and the previous causal probability $P_{i \rightarrow j}^{\text{pre}}$. These variables are available locally at the causing process $i$, however, for $\bar{f}_i$, the process needs to do some bookkeeping.

Note that $\Delta P_{i \rightarrow j}$ is only nonzero, if $x_i$ is nonzero. If we think about the causal relationship, this makes sense. $P_{i \rightarrow j}$ corresponds to the hypothesis, that $i$ is causing $j$. Hence, if we observe the causing observable $i$ to be false, we can not learn anything about its relationship to $j$: If $j$ is also false they could just happen to be both false at the same time. If $j$ is true, something else could be causing it at the moment. However, if the causing observable $i$ is true, $j$ must also be true if the hypothesis of $i$ causing $j$ is true.

Figure 6.13 shows the computed and learned probabilities of the relations of the causal graph of Figure 6.12 based on 2000 samples. The conditional probabilities are computed by the frequency approximation $P(x_j|x_i) \approx \frac{f_{ij}}{\bar{f}_i}$ from the set of all samples in one batch. The right column used the incremental learning rule presented above. Note that from the
Figure 6.13: Computed and learned Bayesian probability of relations based on 2000 samples drawn uniformly from the causal world of graph in Figure 6.12. Axis in all figures are \( x_i \) on the x-axis, and \( x_j \) on the y-axis. The left column shows the connectivity matrix of the graph, the lower plot shows the connectivity matrix with indirect causes added. The ‘Batch computed’ column shows the explicitly computed Bayesian probability \( P(x_j|x_i) \), and the ‘Learned’ column shows the learned probability \( P_{i\rightarrow j} \). The lower row shows the thresholded probabilities at 0.8. Both, computed and learned almost completely represent the true connectivity matrix on the left. The left column shows the correlation of the inputs as control and the correlation thresholded arbitrarily at 0.5. As a comparison the rightmost column shows the correlation between any two nodes. The bottom right figure is the correlation thresholded at 0.8.

samples it is impossible to retrieve the exact causal relations of chains of causes. I.e. in Figure 6.12, observable 2 directly causing observable 7 can not be distinguished from observable 2 causing 4 and 5 and then causing 7, for example. To illustrate that causal relationship is not as simple as the correlation between any two nodes we added the correlation as reference.

The connectivity matrix with included indirect causes is shown in the bottom left corner. Arbitrarily thresholding the probabilities of the computed and learned matrices at 0.8 shows that after 2000 samples the causations have been almost entirely correctly reconstructed. The problem is that from the samples a relation of 8 causing 7 is guessed. This happens because the majority of the samples from Figure 6.12 contain 7 and 8 active together. The only sample testifying for 8 not causing 7 is the sample of the second row, which occurs only with probability 1/11. 9/11 of the samples show situations that suggest 8 causing 7. Indeed, the computed probability of element (8,7) approaches 9/10.
6.6.2 Analysis of the Learning rule

We can compute the statistic average of the learning update $\Delta P_{i\rightarrow j}$ in the case where the learned parameter $P_{i\rightarrow j}$ is the true conditional probability of $P(x_j|x_i)$:

$$\langle \Delta P_{i\rightarrow j} \rangle = \left\langle \frac{x_i}{f_i} \cdot (x_j - P_{i\rightarrow j}) \right\rangle$$

$$= \frac{1}{f_i} \left[ \langle x_i x_j \rangle - \langle x_i \rangle P_{i\rightarrow j} \right]$$

$$= \frac{1}{f_i} \left[ P(x_i, x_j) - P(x_i)P(x_j|x_i) \right]$$

$$= 0. \quad (6.20)$$

Because $x_i$ takes on values 0 or 1, its average $\langle x_i \rangle$ is exactly the probability of occurring in state 1: $P(x_i)$. The last step of (6.20) is done by virtue of Bayes’ theorem $P(x,y) = P(y|x)P(x)$. We see that on average, the learning rule cancels out if $P_{i\rightarrow j} = P(x_j|x_i)$.

We can also look at the variance of the learning update:

$$\langle (\Delta P_{i\rightarrow j} - \langle \Delta P_{i\rightarrow j} \rangle)^2 \rangle = \langle \Delta P^2_{i\rightarrow j} \rangle$$

$$= \left\langle \frac{x_i^2}{f_i^2} (x_j - P_{i\rightarrow j})^2 \right\rangle$$

$$= \frac{1}{f_i^2} \left[ \langle x_i^2 x_j^2 \rangle - 2P_{i\rightarrow j} \langle x_i^2 x_j \rangle + P^2_{i\rightarrow j} \langle x_i^2 \rangle \right]$$

$$= \frac{1}{f_i^2} \left[ \langle x_i x_j \rangle - 2P_{i\rightarrow j} \langle x_i x_j \rangle + P^2_{i\rightarrow j} \langle x_i \rangle \right]$$

$$= \frac{1}{f_i^2} \left[ P(x_i, x_j) - 2P(x_j|x_i)P(x_i, x_j) + P(x_j|x_i)^2 P(x_i) \right]$$

$$= \frac{1}{f_i^2} \left[ P(x_i, x_j) - 2P(x_j|x_i)P(x_i, x_j) + P(x_j|x_i)P(x_i) \right]$$

$$= \frac{1}{f_i^2} \left[ P(x_i, x_j) - 2P(x_j|x_i)P(x_i, x_j) + P(x_j|x_i) \right]$$

$$= \frac{1}{f_i^2} \left[ P(x_i, x_j) - P(x_j|x_i) \right] \cdot (1 - P(x_j|x_i)) . \quad (6.27)$$

Because $x_i$ and $x_j$ are binary variables $\langle x_i^2 \rangle = \langle x_i \rangle$ and thus we can simplify from (6.23) to (6.24). The step (6.25) to (6.26) follows directly from Bayes theorem. The variance (6.27) decays with $f_i^2$, the frequency of an event.
6.6. Connectivity from Observation

occurring at \( i \), and thus converges towards 0. It is therefore safe to say that the learning rule correctly converges towards the actual conditional probability.

From the equation (6.20) we see that the factor \( \frac{1}{f_i} \) does not have an influence on the attractor of \( P_{i \rightarrow j} \). It will have an influence on the variance \( \langle \Delta P^2_{i \rightarrow j} \rangle \) and thus on the convergence. However, if we consider causal networks with only deterministic causes where the actual conditional probabilities \( P(x_j|x_i) \) are either 0 or 1, we can accelerate the convergence by substituting \( 1/f_i \) in (6.19) with a constant small learning rate \( \eta \):

\[
\Delta P_{i \rightarrow j} = \eta x_i (x_j - P_{i \rightarrow j})
\]  

(6.28)

This works especially well for causal cases in which \( P(x_j|x_i) = 1 \) because there, the variance (6.27) approaches 0 even if \( \frac{1}{f_i} \) is substituted by a constant learning rate \( \eta \). However, if \( P(x_j|x_i) = 0 \), the variance is proportional to \( \eta^2 P(x_i, x_j) \). \( \eta \) should therefore be chosen sufficiently small.

6.6.3 Simulation in Network of Processes

We implemented the learning rule (6.28) in a network of processes. We assumed the observable variables to be stationary processes in the network that broadcast their variables as messages into the network. We call these processes sensory variables \( s\# \), where \( \# \) is an enumeration. A sensory process \( s\# \) is a sensor of the world and make its value available to processes within the network. The causal graph is rebuilt inside the network as a model of the world by observables, processes, that can migrate and replicate and connect to one another. Observables are movable processes that occupy network nodes with sensory processes. Each sensory variable needs a corresponding observable if its value should to be processable by the graph of processes inside the network. Observables contain several modules:

**Observable Replicator Module** This module controls which sensory variable this process is observing. The first process is unspecialized until it receives a first broadcast message from a sensory process – it then becomes specialized to observe this particular sensory variable. If a specialized process receives a broadcast message of a variable it is not observing, this module replicates the process and commands the newly born process to observe the unobserved variable. It also sets the migrational preference of its sensory variable to 20.
**Migrate Module** This module (as seen in section 6.4) migrates the process according to its migrational objective function. Observables, for example, migration on top of their observed sensory variable.

**Sensor Module** This module distributes the value of the observable to subscribed destination processes. It also broadcasts the value unspecifically into the network.

**Causation Module** This module keeps track of the values of all the other observables it receives by broadcast from their Sensor Modules and relates itself to them by applying the learning rule (6.28). If a probability of cause $P_{i \rightarrow j}$ between itself ($i$) and another observable $j$ rises above the threshold 0.85 it regards itself as cause of $j$ and adds $j$ to its connections. If $P_{i \rightarrow j}$ drops below 0.8, the connection is removed again. If connected, the CausationModule also sets the migrational preference of this process to 1 to optimize co-localization in processes that are not bound to sensory input.

Figure 6.14 shows the development of the observables. The sensory variables of the graph of Figure 6.12a are distributed in the network from left to right, randomly in the y dimension. a To start off, a single observable (Mother) is placed randomly in the network. The sensory variables randomly alternate between possible states of the world (Figure 6.12a) every two seconds. In b Mother has specialized into observable $o7$, that will be observing sensory variable $s7$. In c through e, the observables replicate, migrate and occupy their corresponding sensory variable.

The observables will send out unaddressed messages with their state. These messages diffuse in the network and are read by other observables. They will immediately start to relate themselves to the other observables and build a table of $P_{i \rightarrow j}$s for all other observables from which they receive messages. Figure 6.15 shows the connectivity after about 300 seconds. Notice that observable $o1$ is connected to $o2$ and $o5$. As discussed in 6.6.1, the learning rule can not distinguish between direct and indirect cause. $o5$ is indirectly caused by $o1$ through $o2$. Eventually all indirectly caused observables will be connected. Note also, that observable $o8$ incorrectly expects to cause $o7$ as can be anticipated from Figure 6.13. Figure 6.16 shows the results of the simulations. The graph structure was correctly recovered by the algorithm.
Figure 6.14: Development of the mode for the world of Figure 6.12. a through e: snapshots at different consecutive times. Observable variables are injected through sensory processes $s1$ through $s9$. a: One process ($Mother$) is inserted into the network. b: This process specializes into the observable $o7$ and then (c through e) replicates into other observables until all sensory processes are occupied.
Figure 6.15: The learned causal connectivity for the world of figure 6.12. 1 through 9: The connections for process 1 through 9.
6.7 Connectivity from Temporal Correlation

In the previous section we extracted causal relations through inference from states observed in the world. We could do this based on the assumption, that at each moment in time, the world is in a consistent state. Just by observing the set of possible states of the world we inferred a model. So far we have discarded the temporal properties of the data in our model construction scheme. However, the temporal component of the data contains a lot of information about the causality of events. The easiest to see:

\( A \Rightarrow B \) If \( A \) consistently occurs slightly before \( B \), \( A \) might very well be the cause of \( B \).

\( A \not\Rightarrow B \) \( B \) can not be caused by \( A \), if \( A \) occurs consistently later in time than \( B \).

From these two facts we can derive a simple temporal learning rule. It has its origins in models on synaptic plasticity and is called spike timing dependent plasticity (STDP)\[6\]. Figure 6.7 shows the learning rule.

\( \Delta W \) is the change in the weight between the presynaptic to postsynaptic cell. If the postsynaptic action potential occurs at the synapse slightly after the presynaptic action potential causes the synapse to fire (in the post region), the weight of the synapse is increased (rule \( A \Rightarrow B \)). If the postsynaptic action potential occurs before the synapse fires due to a presynaptic action potential, the weight is decreased (rule \( A \not\Rightarrow B \)). We can use this exact rule for the inference of causation between observables.

We now exchanged the **Causation Module** with a **STDP Module** in our simulation. Like in section 6.6 processes are trying to learn their relation to one another, but instead of using learning rule (6.28) they now use the
The world needs to properly reflect now the temporal dynamics of the causal relations. Unlike before, where the world just represented the global state at any given moment in time. In the graph of Figure 6.12, this means for example that if observable 1 is active at a given moment in time, it causes a temporal cascade of activity of first 2, then 4 and 5, then 7 and then 8. For STDP to work properly, the characteristic length of the learning rule needs to be in the order of magnitude of the time delay between a cause and its effect. Co-activation of observables that occur outside this window, will not have an effect on their relation. In particular, an indirect cause will take longer to take effect than a direct cause. Therefore, indirect causes will not be captured by an STPD learning rule that is properly tuned. This is good because we do not want indirect causes to be directly related as we had with the causal learning rule (6.28).

Figure 6.18 shows a simulation of the graph from Figure 6.13. The reconstruction of the network is very well. Notice now how indirect causes not correctly not detected. There is, however, a confusion about 6 causing 7. Studying the graph of Figure 6.13 shows that whenever node 3 is active it causes node 6 to be active exactly one transmission delay before node 7, even though caused through a different path. This chain of events creates a sample that speaks for 6 causing 7. Only in the occasion when node 6 is active but not caused by node 3, will there be a sample that testifies for 6 not causing 7. Our algorithm hence assumes causation.
Figure 6.18: The learned relations in the network. a: Theoretical probability matrix as simulated under perfect conditions, excerpt from Figure 6.13. b: Learned relations in the built graph of processes in the network with the STDP learning rule. The STDP almost perfectly reconstructs the graph with no indirect causes. However, it confuses the cause of 6 to 7. c Standard deviations of the learned probabilities. Most fluctuations remain below 0.2.

6.7.1 Prediction and Novelty

We can utilize the causal models we have constructed in the previous subsection to make predictions about the near future. There are many reasons why prediction is very useful. In Nature, it is absolutely essential. For example, both, the hunter and the hunted use their predictive model of the world to plan their hunt or adjust their reaction when surprised by a hunter.

In a more simple case, such a model could be to learn a relationship between an unconditioned stimulus and a reward. In this world, we can learn these relationships inductively: when two events occur simultaneously, repeatedly, we can assume they are related because the time evolution of the world follows deterministic rules and the global organization of the world and its rules don’t change on the scale of our observations.

As seen above, prediction can be very important for control. A predictive controller is a better controller than a purely reactive controller (if its predictions are correct, that is). In our view, control can be regarded very abstractly as the emission of control signals that maintain homeostasis as we have seen in Section 3.3. In this context, a control signal could be an activation of an inhibitory neuron to prevent propagation of a sensory signal that can be anticipated by prediction, or it could be the release of enzymes into the blood to keep a certain concentration constant. Finally, it could be a complicated behavioral action that aims at survival in an inevitable fight.

The example above about the inhibitory neuron has a very clear application: It minimizes information throughput. In information theory we see that to minimize information throughput, we want to use short codes for common events and longer codes for less common events[104]. To expand this principle even further we can say: If A causes B, every time A has
occurred, B will occur (it is infinitely common after A). Hence, we don’t need to encode this predictable event anymore. Predictable events can be reconstructed by the internal model. The transmitting of non-predictable events, i.e. novelties, suffices because only they can not be anticipated from the state of the world just before. The implementation of predictory systems of this kind date back to temporal difference learning [109] and were discovered in the dopamine system of primates [102].

We will distinguish two different types of events: (1) Novelty events are events that are caused by something outside the observable model. (2) Caused events are events that are caused by preceding variables of the world. It will be the task of the model constructor to distinguish novelty and caused events properly. To test our inhibition scheme, we added the Prediction Module to our processes. The STDP Module of the previous subsection already sends out the activity of this process to its (learned) caused processes. The Prediction Module of the receiving process now uses this activity inhibitorily to suppress the activation of an incoming sensory signal received through its Observable Replicator Module. If indeed, its activation was caused by a preceding observable process, it then is correctly inhibited. If there was no preceding observable process active, the sensory signal will fully activate this process, thus signaling a novelty that has occurred. Figure 6.19 shows the result of a simulation of the graph of Figure 6.12. On average, 85% of all events were correctly detected as novelty events or correctly declined as caused events.

6.8 Conclusion

In this chapter we investigated the feasibility of a self-constructing system with replicating processes that are can distribute themselves a network. We demonstrated how to use our developed principles to build predefined graphs in networks of unknown topology. Because we do not allow ourselves global knowledge about the topology of the network, nor the position of the processes, the challenge is to build these structures from the inside out. A smart routing algorithm optimizes routing between distant nodes and implicitly provides a metric in the network that allowed processes to migrate accordingly. Every node of the graph knows its desired local environment (i.e. the nodes it should be connected to). Because processes can replicate, they can develop and maintain the local structure of the graph without global observation. In this study we have not closed the global feedback loops of Section 3.2.4. This would require a more elaborate model of process functions and communication protocols. Next steps
Figure 6.19: Novelty detection in the graph of Figure 6.12. a The performance of each observable in a single trial over 100,000 events. At every observable, the same amount of novelty events were injected. Because events propagate along the causal arcs of the graph, events accumulate from 1 through 8. b The mean performance of each observable measured in correct decision per number of processed events across 5 simulations. For all simulations, the performance was very similar: error bars show standard deviations. c Explanation of cases. A novelty always results in activity, but not every activity is a novelty because it could also have been caused by a preceding observable.
pointing in this direction would be to study the kinds of emergent effects that can be produced through the structure or the function of the embedded graph. Similar to the earlier chapters, it is these effects that would allow to extract a local signal from global structure. As before, eventually, we would like to be able to encode the type of problem we want to solve - the solution should emerge out of the developing structure. But these methods have yet to be discovered.

We also showed how local functional rules and environmentally derived activity can give rise to structure. Either with the causal learning rule (6.19) or with STDP-type learning. Using these local rules, the system is able to rebuild a model of the world just from observing data that is produces by the world. The internal model can then be used to predict future stimuli. This proves to be very useful when minimizing information load: A predictable stimulus does not need to be encoded anymore, it can be anticipated by the internal model. In this scheme, only entirely novel stimuli produce activity. There is a resemblance to temporal difference learning [109] (TD-learning). In TD-learning, conditioned stimuli give rise to an immediate internal reward signals when a future reward can be anticipated. Later, the chain of states that finally leads to the reward, including the external reward at the end of the chain do not produce internal reward signals anymore because they were predicted and thus internally rewarded already.

Although our examples in this thesis are simple, we expect this model to be applicable to more complex problems of real-world type. For example, it could be implemented in peer-to-peer networks to find correlated or caused activity of peers across the network without any a priori assumptions about the structure of the network or the models the agents are subject to. The detection of sequences of actions and novelty allows for an abstraction of high level events that happen in the world. As deterministic sequences of actions can be completely anticipated they contain no more information than the initial trigger that causes them. Thus a sequence of actions can be understood as a symbol for a higher level process. As discussed in Section 3.3.2, using abstract representations, a simpler reasoning machine can be built. The intermediate logic that computes on the abstract states could be built by self-constructing graphs as shown in Section 6.5.
Chapter 7

Discussion

7.1 Conclusion

In the beginning of this thesis, we have posed the question: “Given a self-replicator, what would be the principles of engineered multicellular self-construction”. There are many reasons that justify exploration of self-construction; two of the more prominent ones are the increase of complexity in engineered systems and the incapability to react to unforeseen environmental influences that cause damage to the system or leave it in an undefined state. This thesis demonstrated how these problems could be tackled utilizing self-construction. However, while self-construction offers nice new possibilities, it also brings along new problems: Because the construction process lacks an external outside observer and manipulator, the assembly has to be achieved from the inside out.

The trick presented in this thesis was to utilize emergent properties of the compound system that signal the qualities of the global structure to local components. We have proposed a set of basic modules that can be used to build higher level structures: For example, the AxisReactor allows us to create global gradients across an entire cell population to form two partitions. Other local reactors and conditions help us to limit the size of the population. These modules can be regarded as the basic building blocks of self-constructing systems, from which more detailed structures can be built. Using these modules and the feedback of local signals to processes, we have described a possible mechanism for hierarchical construction of non-trivial structures starting from one cell. The previously set up structure guides the further development.

Because the developing system is in contact with its environment, it can use environmentally derived activity to guide its development. Using this
mechanism a self-constructing system can adapt to previously unknown environmental conditions. As an example we saw how an organism could develop to a specific sensorimotor connectivity so as to optimize its received reward. However, because development allows many different possibilities, the search space of the learning algorithm is usually too large. As a solution we proposed lowering the dimensionality of the search space by limiting the possible developmental routes a system can take, for example by introducing global dynamics that lead to a small set of stable attractors. The adaptation then reduces to the choice of the correct attractor.

As an extreme case, in the context of networks of processes, we discussed the complete extraction of an environmental model from observation of the world. We proposed a scheme whereby processes can self-replicate, distribute and connect themselves so as to optimally predict the signals emerging from the environment. The prediction mechanism gives rise to a symbolization of the complex input, and allows for distinction between novelty and cause.

However, inside-out construction poses new question on the design of systems. Our traditional top-down design and engineering mechanisms do not grip anymore. Thus, we have to seek for novel design strategies.

7.2 Design Methods

For its design, previous work on self-construction has relied heavily on genetic algorithms to discover suitable assembly instructions. Genetic algorithms are useful for finding solutions in a large search space, but we consider that these algorithms are an inappropriate foundation for an effective technology of self-construction for the following reasons.

Firstly, the use of genetic algorithms is only feasible when fitness can be evaluated quickly. They are less suitable for configuring self-constructing systems where a large fraction of time is required for the development of the system before it can be evaluated. In particular, a trial and selection scheme with real hardware in a real environment would also be economically unfeasible. The search process could be sped up by simulation [73]: But this approach raises the additional problem of the reality gap, the imprecisions in the mapping between the simulated universe and reality.

Secondly, a technology is usually task orientated. That is, we require that the self-organized system possess some specific physical characteristics and functional competences, which will enable the phenotype to perform (economically) some target task. Ideally, we would like to specify the characteristics, competences, and tasks explicitly using a high-level design
Design Methods

language that is able to generate the Description that will be inserted into the progenitor cell (or ‘stem cell’) of the self-constructing system. Our overall approach is to constrain the developmental process to unfold in such a way that the earliest cell populations interact with one another to provide a physical infrastructure, within which the later more task-related populations can be configured.

Figure 4.5 shows how our organism unfolds according to a specific plan that can be encoded as a kind of state machine. This formal description as state machine raises the exciting possibility that self-assembling organisms could in future be designed efficiently, rather than relying on expensive search. As is known for algorithmic compression, however, it is a highly non-trivial problem to find the optimally compressed description of a pattern. Just as this is a hard problem in computer science, in the design of self-constructing systems it may require the experience, creativity and ingenuity of a human brain or the time and selective pressure of Darwinian evolution to create an appropriate description for a specific structure. To this point, it is not clear whether automated design is possible. Self-construction thus brings along novel requirements for the design and engineering of such systems. As Drexler points out[34] in the context of nano-assembly, we may need molecular and design compilers that aid the design and construction. In our proposed scheme of self-construction, the construction is carried out by the system itself following an indirect description that is non-trivially related to its target structure and function.

Compilers essentially replace strings specified in a given grammar by a low level code description through the substitution of keywords into low level code snippets. Once expanded into a low level code, it can be optimized locally. However, the process of optimization usually removes the reversibility of the mapping between high level and low level description, because it alters the local structure to deviate from the snippets composition. The resulting low level description can now again be passed to a next stage compiler that creates a description even lower level. In the context of self-construction, an optimization stage could be carried out throughout development and wouldn’t have to be done at design time. If the correct local processes are found to carry out the optimization, the design and engineering of a self-constructing system will therefore be reduced to the specification of abstract structures – the implementation details should be resolved by the construction process itself. In the future the bulk of the effort will therefore be put into the design of the design infrastructure of such systems, rather than into its construction.
7.3 The Future of Self-Construction

Principles of self-construction and -repair are relevant not only for biology, but also for the design and construction of advanced software, machines, and buildings. Distributed sensitivity to its very own structure and local goal directed behavior of components will find applications everywhere where systems are embodied in a real world and interfaces cannot be well defined a priori.

7.3.1 in Mechanical Technology

A physical implementation of self-constructing systems in mechanical technology seems still far ahead. However, recent advances in automated design and construction\cite{73} and in the field of self-assembling robots\cite{129} show that in principle, mechanical self-construction is possible. But it will require much more sophisticated and miniaturized construction mechanisms to be able to construct useful systems.

Another path to self-construction in mechanical technology leads through the field of passive, mechanical self-assembly. Gracias et al.\cite{49} demonstrated the self-assembly of a 3D circuit and Zheng et al.\cite{128} show that the self-assembly of functional structures is possible. Although these preliminary results look promising, self-construction of complex machinery will require an active component in the assembly process to build very specific structures depending on conditions derived from the environment. In this context, to me, passive self-assembly has a feel to it like life before Nature's invention of RNA-polymerase.

7.3.2 in Biology

Biological implementation seems to be in reach sooner: Cells have been stripped down to their essential components and genes necessary to survive\cite{59}; and there are chemical models for biological self-assembling multicellular systems\cite{60}. Very conveniently, a biological cell already contains the crucial components for self-construction: RNA-polymerase in the role of the constructor, DNA in the role of the code, a cell membrane that makes a cell an closed object and a mechanism for self-replication through cell cleavage.

In the emerging field of “synthetic biology”, attempts are being made to catalogue simple gene sequences and their functions in order to later assemble them into complex circuits in synthesized DNA\cite{5}. Figure 7.3.2
7.3. The Future of Self-Construction

Figure 7.1: These four bottles contain the substrate for synthesizing the four base nucleic acids Thymine, Cytocine, Guanine, Adenine (from left to right). Current technology permits DNA-synthesis of any sequence of DNA given an electronic specification in the form of a string of A, C, G and Ts. (Image, courtesy of Drew Endy, MIT)

shows four bottles of substrates that are used to synthesize arbitrary strands of DNA from a digital specification. There are many companies in the world that synthesize any DNA on request. An order usually ships within two to four weeks and costs less than $5000. It is becoming very feasible now to start experimenting with engineered self-construction of organic systems. Recently genetic engineering has achieved successes underlining this direction[22, 113]. In particular, the synthesis of a multicellular organic system with programmed pattern formation has been achieved[14].

We have been careful not to violate locality of interactions in our study. Also, the internal cellular mechanics very much resemble those of a real biological cell. We thus expect the principles laid out in this thesis to translate rather straightforward to a real organic implementation.

7.3.3 in Software Design

As we have discussed in Section 2.3, novel paradigms of software design (such as Extreme Programming)[15] also aim at closer feedback cycles between design, construction and testing. More effort is now being put in advanced integrated development environments (IDEs) that are aiding the development of software to a great extent by providing functionality such as automated refactoring and other algorithmically achievable features. Syntactic errors can usually be correctly detected and fixed. Language specifics are now being handled by the IDE and protocols are defined in universal
design patterns[45]. The work that is left to the programmer is merely the design of the overall system, i.e. how components are related, and the implementation of very specific functionality. From this perspective, software engineering boils down to plugging the right components together and adding some low level functionality at the right places.

Self-construction would allow software systems to create objects and their encapsulated functionality at will and relate them to the previously set up object structure. Simple rules that describe which objects should be instantiated based on the current state of the system and how they should be related to the previous structure will have to be discovered. One of the difficulties that we see in this endeavor will be the discrete quality of objects: Adding an object and putting it to function in a previously set up structure might greatly change the resulting function of the system. It will be very hard to anticipate what actions to take and what their resulting behavior will be. Thus we suggest to keep the function of a single object very small such that incremental changes in structure will result in incremental changes in function. This will require the assembly of many small objects rather than just a few specialized big ones. As we have done in the context of artificial chemistries, also in software systems, one would have to find small interactions of objects that lead to global phenomena that can be used to guide further development.
Appendix A

Analysis of the Behavior of a Perfect Braitenberg Vehicle

The oscillating behavior of the organism along a path of good is a property of all Braitenberg vehicles. This can be seen by looking at a simplified environment in which the food concentration is constant along the $y$-axis and given by some function $f(x)$ along the $x$-axis (see Figure A.1). The configuration space of the vehicle is given by the triple $(x, y, \theta)$ that contains the vehicle’s $x$, $y$ coordinate and its angle $\theta$ in the environment. By definition, when $\theta = 0$, the vehicle points to the right in the direction of the increasing $x$-axis. A simplified Braitenberg vehicle obeys the dynamics

$$\dot{x} = f(x) \cos(\theta) \quad (A.1)$$
$$\dot{y} = f(x) \sin(\theta) \quad (A.2)$$
$$\dot{\theta} = -f'(x) \sin(\theta). \quad (A.3)$$

Equation (A.3) is given by the fact that the angle $\theta$ changes based on the gradient of the food across the sensory input. The gradient is proportional to $\sin(\theta)$. This can be seen by looking at a vehicle pointing right and thus not facing any gradient while a vehicle pointing up is seeing the whole food gradient $f(x)$ at location $x$.

Because $y$ does not enter the equations of the other two variables $\dot{x}$ and $\dot{\theta}$ the dynamics can be studied by looking at the configurations space $(x, \theta)$. Figure A.1 shows the flow of the dynamics subject to the food $f(x) = \cos^2(x)$. There are singularities at the points

$$(x, \theta) = (i \cdot \pi, \frac{\pi}{2} + j \cdot \pi), \quad (A.4)$$

for $i, j \in \mathbb{Z}$. These are exactly the configurations where the vehicle is sitting on top of the food hill and pointing either up or downwards along the crest.
Analysis of the Behavior of a Perfect Braitenberg Vehicle

Figure A.1: Left: Simply laid out food for analysis. Right: Flow of dynamics for the given food. Axes are $x$ and $\theta$. TODO: add arrows pointing in theta direction to y-axis. substitute theta by $\theta$ in figure. add food profile to x-axis.

Of the hill. Even though $x$ and $\theta$ are not changing here, the vehicle is now moving at maximum speed $f(x)$ in the $y$-direction.

We can see that $\dot{x}$ and $\dot{\theta}$ are partial derivatives of the same function $u(x, \theta)$:

$$\dot{x} = f(x) \cos(\theta) = \frac{\partial}{\partial \theta} u(x, \theta) \quad (A.5)$$
$$\dot{\theta} = -f'(x) \sin(\theta) = -\frac{\partial}{\partial x} u(x, \theta), \quad (A.6)$$

where $u(x, \theta)$ is

$$u(x, \theta) = f(x) \sin(x). \quad (A.7)$$

In the Hamiltonian formalism it is seen that a dynamical system obeying (A.5) and (A.6) is conserving the quantity $u(x, \theta)$ along its flux. This can also be verified by explicit calculation:

$$\frac{d}{dt} u(x) = \frac{d}{dt} f(x) \sin(\theta)$$
$$= f'(x) \sin(\theta) \dot{x} + f(x) \cos(\theta) \dot{\theta}$$
$$= f'(x) \sin(\theta) f(x) \cos(\theta) - f(x) \cos(\theta) f'(x) \sin(\theta)$$
$$= 0. \quad (A.8)$$

But $f(x) \sin(\theta)$ turns out to be exactly the velocity in the $y$ direction $\dot{y}$. Hence we know that a Braitenberg vehicle will be moving with constant speed in the $y$ direction depending entirely on the initial conditions. Because the flow of the dynamics as seen in figure (A.1) does not have a basin of attraction for the point attractors (A.4), a vehicle will always remain on
Figure A.2: The orbit of a vehicle integrated with sloppy integration of vectorfield. The circle depicts the starting point of the path. It converges towards a limit cycle with low \( y \) speed. Its orbit around the singularity, but will never be drawn into it. This means that a vehicle will always oscillate around the hill of the food.

In our simulations we integrated the dynamics through a simple 1-step integration with an integration constant \( \tau > 0 \):

\[
\begin{align*}
    x^{t+1} &= x^t + \tau \cdot \dot{x}(x^t, y^t, \theta^t) \\
    y^{t+1} &= y^t + \tau \cdot \dot{y}(x^t, y^t, \theta^t) \\
    \theta^{t+1} &= \theta^t + \tau \cdot \dot{\theta}(x^t, y^t, \theta^t).
\end{align*}
\]

(A.9) (A.10) (A.11)

Using these updates for the trace of the vehicle, the behavior is not entirely satisfactory: Instead of oscillating around the food, or even better converging towards the singularity on the maximum food, the vehicle escapes the initial cycle with \( \dot{y} \) constantly decreasing and converges towards the limit cycle with \( \dot{y} = 0 \) (Figure A.2).
Appendix B

Description Code

B.1 XML Code

This appendix contains the complete description code in XML that was used to generate the organism of chapter 4. It corresponds to Table 4.1 and to the values listed in the previous section of this appendix (B.2).

B.1.1 Organism DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT code (gene+)>  
<!ATTLIST gene (regulator,action)>  
<!ATTLIST gene name CDATA "">  
<!ATTLIST gene type CDATA "">  
<!ELEMENT regulator (condition*)>  
<!ELEMENT condition EMPTY>  
<!ATTLIST condition type (ON,OFF) "ON"  
chem NMTOKEN "0"  
thresh NMTOKEN "0">  
<!ELEMENT action (factory*)>  
<!ELEMENT factory (argument*)>  
<!ATTLIST factory name CDATA "">  
<!ELEMENT argument EMPTY>  
<!ATTLIST argument n CDATA "">  
<!ATTLIST argument n CDATA #REQUIRED  
value NMTOKEN #REQUIRED>
```
B.1.2 Organism Code

code.xml

<!DOCTYPE code SYSTEM "code.dtd">

code

<gene name="B0" type="0">
  <regulator>
    <condition chem="18" thresh="1" type="ON"/>
    <condition chem="0" thresh="0" type="ON"/>
    <condition chem="0" thresh="0.8" type="OFF"/>
  </regulator>
  <action>
  <factory name="DivideCompetence" />
  <factory name="MigrateCompetence">
    <argument n="1" value="1"/>
    <argument n="2" value="-1"/>
  </factory>
  <factory name="SourceReactor">
    <argument n="1" value="0"/>
    <argument n="2" value="0.2"/>
  </factory>
  <factory name="ConstReactor">
    <argument n="1" value="18"/>
    <argument n="2" value="2.1"/>
    <argument n="3" value=".1"/>
  </factory>
  <factory name="DecayReactor">
    <argument n="1" value="18"/>
    <argument n="2" value="23"/>
    <argument n="3" value=".08"/>
  </factory>
  </action>
</gene>

<gene name="C" type="1">
  <regulator>
    <condition chem="18" thresh="1" type="ON"/>
    <condition chem="0" thresh=".8" type="ON"/>
    <condition chem="1" thresh="10.5" type="OFF"/>
  </regulator>
  <action>
  <factory name="AxisReactor">
    <argument n="1" value="6"/>
    <argument n="2" value="7"/>
    <argument n="3" value="8"/>
    <argument n="4" value="9"/>
    <argument n="5" value=".13"/>
    <argument n="6" value=".05"/>
  </factory>
  </action>
</gene>
<factory name="MigrateCompetence">
    <argument n="1" value=""/>
    <argument n="2" value="-1"/>
</factory>

<factory name="SourceReactor">
    <argument n="1" value=""/>
    <argument n="2" value=".2"/>
</factory>

<factory name="ConstReactor">
    <argument n="1" value="19"/>
    <argument n="2" value="2.1"/>
    <argument n="3" value=".1"/>
</factory>

<factory name="DecayReactor">
    <argument n="1" value="18"/>
    <argument n="2" value="23"/>
    <argument n="3" value=".05"/>
</factory>
</action>
</gene>

<gene name="BM" type="2">
    <regulator>
        <condition chem="18" thresh="1" type="ON"/>
        <condition chem="0" thresh=".8" type="ON"/>
        <condition chem="6" thresh="1.8" type="ON"/>
        <condition chem="2" thresh=".3" type="OFF"/>
    </regulator>
    <action>
        <factory name="MigrateCompetence"/>
        <factory name="MigrateCompetence">
            <argument n="1" value="6"/>
            <argument n="2" value="7"/>
        </factory>
        <factory name="SourceReactor">
            <argument n="1" value="2"/>
            <argument n="2" value=".2"/>
        </factory>
        <factory name="ConstReactor">
            <argument n="1" value="20"/>
            <argument n="2" value="2.1"/>
            <argument n="3" value=".1"/>
        </factory>
        <factory name="DecayReactor">
            <argument n="1" value="18"/>
            <argument n="2" value="23"/>
            <argument n="3" value=".05"/>
        </factory>
    </action>
</gene>
<gene name="BS" type="3">
  <regulator>
    <condition chem="18" thresh="1" type="ON" />
    <condition chem="0" thresh=".8" type="ON" />
    <condition chem="7" thresh="1.8" type="ON" />
    <condition chem="3" thresh=".3" type="OFF" />
  </regulator>
  <action>
    <factory name="DivideCompetence" />
    <factory name="MigrateCompetence">
      <argument n="1" value="7"/>
      <argument n="2" value="6"/>
    </factory>
    <factory name="SourceReactor">
      <argument n="1" value="3"/>
      <argument n="2" value=".2"/>
    </factory>
    <factory name="ConstReactor">
      <argument n="1" value="21"/>
      <argument n="2" value="2.1"/>
      <argument n="3" value=".1"/>
    </factory>
    <factory name="DecayReactor">
      <argument n="1" value="18"/>
      <argument n="2" value="23"/>
      <argument n="3" value=".05"/>
    </factory>
  </action>
</gene>

<gene name="M" type="4">
  <regulator>
    <condition chem="20" thresh="1" type="ON" />
    <condition chem="0" thresh=".8" type="ON" />
    <condition chem="6" thresh="1.8" type="ON" />
    <condition chem="2" thresh=".3" type="ON" />
    <condition chem="4" thresh="3.5" type="OFF" />
  </regulator>
  <action>
    <factory name="AxisReactor">
      <argument n="1" value="10"/>
      <argument n="2" value="11"/>
      <argument n="3" value="12"/>
      <argument n="4" value="13"/>
      <argument n="5" value=".13"/>
      <argument n="6" value=".1"/>
      <argument n="7" value="3"/>
    </factory>
  </action>
</gene>
<argument n="8" value=".02"/>
</factory>

<factory name="ExcitatoryMotorCompetence">
  <argument n="1" value="5" />
  <argument n="2" value="14" />
  <argument n="3" value="15" />
</factory>

<factory name="MigrateCompetence">
  <argument n="1" value="6" />
  <argument n="2" value="7" />
</factory>

<factory name="SourceReactor">
  <argument n="1" value="4" />
  <Argument n="2" value=".2" />
</factory>

<factory name="ConstReactor">
  <argument n="1" value="22" />
  <argument n="2" value="2.1" />
  <argument n="3" value=".1" />
</factory>

<factory name="DecayReactor">
  <argument n="1" value="18" />
  <argument n="2" value="23" />
  <argument n="3" value=".05" />
</factory>

</action>
</gene>
<gene name="S" type="5">
  <regulator>
    <condition chem="21" thresh="1" type="ON" />
    <condition chem="0" thresh=".8" type="ON" />
    <condition chem="7" thresh="1.8" type="ON" />
    <condition chem="3" thresh=".3" type="ON" />
    <condition chem="5" thresh="3.5" type="OFF" />
  </regulator>

  <factory name="InterAxisReactor">
    <argument n="1" value="14" />
    <argument n="2" value="15" />
    <argument n="3" value="16" />
    <argument n="4" value="17" />
    <argument n="5" value="10" />
    <argument n="6" value="11" />
    <argument n="7" value=".13" />
    <argument n="8" value=".1" />
    <argument n="9" value="3" />
    <argument n="10" value=".02" />
  </factory>

  <factory name="SensorCompetence"
</factory>
<factory name="adapt.AxonMotorCompetence">
    <argument n="1" value="10"/>
    <argument n="2" value="11"/>
    <argument n="3" value="14"/>
    <argument n="4" value="15"/>
    <argument n="5" value=".5"/>
    <argument n="6" value="4"/>
    <argument n="7" value="6"/>
</factory>
<factory name="SensorCompetence">
    <argument n="1" value="0"/>
</factory>
<factory name="AxonateCompetence">
    <argument n="1" value="10"/>
    <argument n="2" value="11"/>
    <argument n="3" value="15"/>
    <argument n="4" value="14"/>
</factory>
<factory name="MigrateCompetence">
    <argument n="1" value="7"/>
    <argument n="2" value="6"/>
</factory>
<factory name="SourceReactor">
    <argument n="1" value="5"/>
    <argument n="2" value=".2"/>
</factory>
<factory name="ConstReactor">
    <argument n="1" value="23"/>
    <argument n="2" value="2.1"/>
    <argument n="3" value=".1"/>
</factory>
<factory name="DecayReactor">
    <argument n="1" value="18"/>
    <argument n="2" value="23"/>
    <argument n="3" value=".05"/>
</factory>
</action>
</gene>
</code>
B.1.3 Environment

```xml
<!DOCTYPE root>
<environment>
  <chemistry nchem="24"/>
  <substrate dimX="30" dimY="30" diffusion=".7">
    <diffusion chem="6" coefficient=".1"/> <!-- g_1C -->
    <diffusion chem="7" coefficient=".1"/> <!-- g_2C -->
    <diffusion chem="8" coefficient=".85"/> <!-- s_1C -->
    <diffusion chem="9" coefficient=".85"/> <!-- s_2C -->
    <diffusion chem="10" coefficient=".08"/> <!-- g_1S -->
    <diffusion chem="11" coefficient=".08"/> <!-- g_2S -->
    <diffusion chem="12" coefficient=".65"/> <!-- s_1S -->
    <diffusion chem="13" coefficient=".65"/> <!-- s_2S -->
    <diffusion chem="14" coefficient=".08"/> <!-- g_1M -->
    <diffusion chem="15" coefficient=".08"/> <!-- g_2M -->
    <diffusion chem="16" coefficient=".65"/> <!-- s_1M -->
    <diffusion chem="17" coefficient=".65"/> <!-- s_2M -->
  </substrate>
  <membrane>
    <diffusion chem="0" coefficient=".8"/> <!-- b_0 -->
    <diffusion chem="1" coefficient=".8"/> <!-- c -->
    <diffusion chem="2" coefficient=".8"/> <!-- b_M -->
    <diffusion chem="3" coefficient=".8"/> <!-- b_S -->
    <diffusion chem="4" coefficient=".8"/> <!-- m -->
    <diffusion chem="5" coefficient=".8"/> <!-- s -->
    <diffusion chem="6" coefficient=".025"/> <!-- g_1C -->
    <diffusion chem="7" coefficient=".025"/> <!-- g_2C -->
    <diffusion chem="8" coefficient=".6"/> <!-- s_1C -->
    <diffusion chem="9" coefficient=".6"/> <!-- s_2C -->
    <diffusion chem="10" coefficient=".03"/> <!-- g_1S -->
    <diffusion chem="11" coefficient=".03"/> <!-- g_2S -->
    <diffusion chem="12" coefficient=".55"/> <!-- s_1S -->
    <diffusion chem="13" coefficient=".55"/> <!-- s_2S -->
    <diffusion chem="14" coefficient=".03"/> <!-- g_1M -->
    <diffusion chem="15" coefficient=".03"/> <!-- g_2M -->
    <diffusion chem="16" coefficient=".55"/> <!-- s_1M -->
    <diffusion chem="17" coefficient=".55"/> <!-- s_2M -->
  </membrane>
</environment>
```
B.2 Parameter Values

Thresholds

In our simulation we used the following thresholds for the values in the above table:

\[
\begin{array}{cc}
\eta & 1.0 \\
\hat{b}_0 & 0.8 \\
\hat{c} & 10.5 \\
\hat{b}_M & 0.3 \\
\end{array}
\begin{array}{cc}
b_S & 0.3 \\
\hat{m} & 3.5 \\
\bar{s} & 3.5 \\
\theta & 1.8 \\
\end{array}
\]

Diffusion coefficients

<table>
<thead>
<tr>
<th>Morphogen</th>
<th>Membrane</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>( b_0 )</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>( \varepsilon )</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>( b_M )</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>( b_S )</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>( b_0 )</td>
<td>0.8</td>
<td>0.7</td>
</tr>
<tr>
<td>( c )</td>
<td>0.8</td>
<td>0.7</td>
</tr>
<tr>
<td>( b_M )</td>
<td>0.8</td>
<td>0.7</td>
</tr>
<tr>
<td>( b_S )</td>
<td>0.8</td>
<td>0.7</td>
</tr>
<tr>
<td>( m )</td>
<td>0.8</td>
<td>0.7</td>
</tr>
<tr>
<td>( s )</td>
<td>0.8</td>
<td>0.7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Morphogen</th>
<th>Membrane</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>( g_{C1} )</td>
<td>0.025</td>
<td>0.10</td>
</tr>
<tr>
<td>( g_{C2} )</td>
<td>0.025</td>
<td>0.10</td>
</tr>
<tr>
<td>( s_{C1} )</td>
<td>0.600</td>
<td>0.85</td>
</tr>
<tr>
<td>( s_{C2} )</td>
<td>0.600</td>
<td>0.85</td>
</tr>
<tr>
<td>( g_{M1} )</td>
<td>0.030</td>
<td>0.08</td>
</tr>
<tr>
<td>( g_{M2} )</td>
<td>0.030</td>
<td>0.08</td>
</tr>
<tr>
<td>( s_{M1} )</td>
<td>0.550</td>
<td>0.65</td>
</tr>
<tr>
<td>( s_{M2} )</td>
<td>0.550</td>
<td>0.65</td>
</tr>
<tr>
<td>( g_{S1} )</td>
<td>0.030</td>
<td>0.08</td>
</tr>
<tr>
<td>( g_{S2} )</td>
<td>0.030</td>
<td>0.08</td>
</tr>
<tr>
<td>( s_{S1} )</td>
<td>0.550</td>
<td>0.65</td>
</tr>
<tr>
<td>( s_{S2} )</td>
<td>0.550</td>
<td>0.65</td>
</tr>
</tbody>
</table>

The diffusion coefficients for all morphogens used in the simulation. The Membrane column specifies the diffusion coefficients of the morphogen through the cell membrane into the local environment. The Environment column specifies the diffusion coefficient on the lattice of the local environment.

**AxisReactor cell type** C: \( c = 0.13, \gamma = 0.05, a = 1, \alpha = 0.02 \)

**AxisReactor cell type** M: \( c = 0.13, \gamma = 0.1, a = 1, \alpha = 0.02 \)

**InterAxisReactor cell type** S: \( c = 0.13, \gamma = 0.1, a = 1, \alpha = 0.02, \delta = 0.6 \)

**SourceReactor** The rates \( r \) for the SourceReactors are the same for all cell types: \( r = 0.2 \)

**ConstReactor** The gain \( \alpha \) for the ConstReactors are the same for all cell types: \( \alpha = 0.1 \)
**MigrateCompetence**  See section MigrateCompetence for a description of the parameters: $\epsilon = 4$, $\lambda = 50$.

The part of the cell’s free energy function depending on the chemical configuration is determined by the following formula:

$$J_p(c_r) = 30 \cdot \langle c_r, c_a \rangle,$$

where $c_r$ is the chemical configuration at node $r$, $\langle \cdot, \cdot \rangle$ is the inner product and $c_a$ is a vector containing the affinities to the specific chemicals, $c_a$ contains $+1$ and $-1$ depending on the cell’s preference to be attracted or repulsed by the corresponding chemical. This is specified in the Description code.

The temperature for the MigrateCompetences is 5. The temperature for the AxonCompetence is 0.1.
Bibliography


[4] ESA website, Mars Express section. www.esa.int. 1


141


Curriculum Vitae

Fabian Roth
born 25.07.1975 in Zürich
Swiss citizen
married

Education
2002 - 2006 PhD under the supervision of Prof. Dr. Rodney J. Douglas at the Institut für Neuroinformatik ETHZ/UNIZ, Switzerland
2000 Diploma thesis at the Centre for Quantum Computation, University of Oxford, UK
1995 - 2000 Diploma course Mathematics with emphasis on theoretical physics at ETHZ Switzerland
1992 - 1993 Highschool student exchange year in Madras, Oregon, USA
1991 - 1995 Kantonsschule Baden

Work
2003 - 2006 Production, design and development of 8 Mobile Games with J2ME technology for Verein Swiss Recycling
2001 - 2002 Java programming for the project *Ada, the Intelligent Space* at INI, ETHZ/UNIZH
2001 Software design and developement in Java for Pfister + Partner System Engineering AG, Rotkreuz
2000 - 2001 System administration and planning of the GNU/Linux server subsytem for T-Online.ch, Uster
1999 - 2000 Teaching assistance at the Institute of Mathematics ETHZ
1999 Teaching Linux system administration courses at the Volkshochschule Zürich
1995 - 2001 Cofounding and working at iotalab GmbH with four partners. iotalab GmbH developed dynamic webapplications and content management systems.

Honors and Awards

- 3rd price in the Playstation 2 Linux VU Assembler Coding Contest 2003 for the demo of a simulated water bubble and lightray refraction.
- Honorary Mention at the Switch Innovation Award 2006 for a project on learning mobile agents ‘Creets’.
Selected Publications

