AMORPHOUS MEMBRANE BLENDING:

FROM REGULAR TO IRREGULAR CELLULAR COMPUTING MACHINES

THÈSE N^{O} 2925 (2004)

PRÉSENTÉE À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS

Institut des systèmes informatiques et multimédias

SECTION D'INFORMATIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Christof TEUSCHER

ingénieur informaticien diplômé EPF
de nationalité suise et originaire de Thoune (BE)

acceptée sur proposition du jury:

Prof. D. Mange, directeur de thèse
Prof. D. Floreano, rapporteur
Prof. M. Sipper, rapporteur
Prof. M. Tomassini, rapporteur

Lausanne, EPFL
2004

To Lalitpuri

ललितपुरी

LaTeX Typesetting

christof@teuscher.ch
www.teuscher.ch/christof

Abstract

The von Neumann architecture was first expressed in 1945 and has largely dominated in many variants and refinements computer science for more than half a century. Alternative architectures always occupied a marginal place only, despite a growing need for new concepts and paradigms in computer science.

Biologically-inspired engineering applies biological concepts to the design of novel computing machines and algorithms. This can lead to the creation of new machines, endowed with properties usually associated with the living world: adaptation, evolution, growth and development, fault-tolerance, self-replication or cloning, reproduction, etc. Most of these approaches are based on well established theories such as artificial neural networks, evolutionary algorithms, and cellular automata.

The work presented in this thesis takes an alternative path and proposes concepts for novel and unconventional biologically-inspired machines. The approach is mainly motivated by the insight that tomorrow's computational substrates and environments might be very different from what we know today. Some of tomorrow's computers might be embedded in the paint that covers your desk or printed on a sheet of paper by means of a special ink. Most of such pervasive computing concepts have some common elements: (1) the computer's basic elements are very simple, identical, and available in a huge number, (2) the interactions between the elements are purely local, (3) the elements as well as the interconnections are unreliable, and (4) there is no global control mechanism available.

This thesis is mainly based on the unification of the following three domains of research: (1) amorphous computing, (2) membrane systems, and (3) blending.

An amorphous computer is a massive parallel machine made up of myr-

iads of simple, unreliable, and identical elements, distributed randomly on a surface and interconnected locally by unreliable connections. Membrane systems are theoretical models inspired by biochemistry-based on regions bounded by membranes. The hierarchical membrane structures contain artificial chemistries, consisting in objects and reactions, which allow to do computations. Blending is a framework of cognitive science which tries to explain how we deal with mental concepts and how creative thinking emerges.

First, an introduction of traditional bio-inspired machines and hardware is provided. This part also includes the presentation of a first implementation of a membrane system on reconfigurable hardware and a description of the cellular automata machine entitled BioWall, with its applications. Random boolean networks as well as several theoretical considerations and practical results are then used to introduce irregular computational structures.

The C-Blending approach represents an novel computational blending method intended for membrane systems and artificial chemistries. In order to implement membrane systems on amorphous computers, the *Circuit Amorphous Computer* as well as special membrane systems, termed aP and aB membrane systems, are proposed. The ultimate concept proposed and studied consists in a unification of membrane systems, amorphous computers, and computational C-Blending.

The unification of the three concepts results in several interesting properties. The cellular structures allow to create dynamical hierarchies and growing systems whereas the artificial chemistries represent an ideal mean to compute on the potentially imperfect and irregular hardware of an amorphous computer. Finally, the computational blending proposed describes an inventive method to create, organize, and adapt membrane systems.

The characteristics and limits of the concepts proposed are analyzed and validated using various examples and toy applications. The thesis concludes with the definition of the *Circuit Amorphous Computer* and the *Amorphon* architecture, which might constitute the minimal element of tomorrow's computing machines.

Version abrégée

Depuis les débuts de l'informatique moderne, la quasi totalité des ordinateurs sont conçus selon l'architecture de von Neumann, présentée en 1945. Les peu nombreuses architectures alternatives n'ont occupé qu'une place marginale parmi les systèmes informatiques, malgré un besoin croissant de nouveaux concepts et paradigmes.

Les approches théoriques et machines informatiques dites bio-inspirées s'inspirent des systèmes biologiques pour se doter de propriétés originales, propres aux seuls êtres vivants: adaptation, apprentissage, évolution, croissance, tolérance aux pannes, autoréparation, etc. Ces machines se basent principalement sur des approches bien établies et connues comme les réseaux de neurones artificiels, les algorithmes évolutifs et les automates cellulaires.

La présente thèse vise à prendre une direction originale en proposant des concepts pour de nouvelles machines bio-inspirées. La principale motivation pour cette voie alternative vient du fait que le substrat et l'environnement de certains futures ordinateurs sera très différent de ce que nous connaissons aujourd'hui. Un ordinateur de demain sera peut-être intégré dans la peinture qui couvre votre table ou pourra être imprimé sur une feuille de papier à l'aide d'une encre spéciale. Ces nouveaux substrats ont tous plusieurs éléments en commun: (1) les éléments de base de l'ordinateur sont simples, identiques, et présents en très grand nombre, (2) les interactions entre ces éléments sont purement locales, (3) les éléments ainsi que les interconnexions ne sont pas fiables, et (4) il n'y a pas de contrôle global.

Cette thèse se base principalement sur trois domaines de recherche: (1) les ordinateurs amorphes, (2) les systèmes membranaires et (3) le *blending*.

Les ordinateurs amorphes sont des machines massivement parallèles constituées d'éléments simples, potentiellement imparfaits, distribués aléatoirement sur une surface, et interconnectés de manière local et aléatoire. Un

système membranaire est un modèle théorique — inspiré par la biochimie — basé sur des régions définies par des membranes qui contiennent des objets effectuant du calcul selon des règles prédéfinies. Le modèle représente ainsi une sorte de chimie artificielle. Le *blending*, théorie provenant des sciences cognitives, essaie d'expliquer comment nous traitons les concepts et comment nos pensées créatives émergent.

La thèse commence par une introduction aux machines et matériel bio-inspirés, incluant une première réalisation de systèmes membranaires sur une structure reconfigurable régulière, ainsi que la présentation de la machine cellulaire BioWall et ses différentes applications. Les réseaux binaires aléatoires ainsi que plusieurs plusieurs considérations et résultats théoriques autour de ce modèle sont utilisés pour introduire et motiver des structures computationelles aléatoires et irrégulieres.

Le C-Blending présente une méthode originale d'application des principes du *blending* à des systèmes membranaires et à des chimies artificielles. Pour réaliser des systèmes membranaires sur un ordinateur amorphe, le *Circuit Amorphous Computer* ainsi que les systèmes membranaires de type aP et aB sont détaillés. L'ultime concept proposé consiste en une réalisation unifiée des systèmes membranaires aP et aB — organisés et adaptés par la méthode du C-Blending — sur un ordinateur amorphe.

L'unification des trois concepts présente plusieurs propriétés intéressantes. Les structures cellulaires permettent la création de hiérarchies dynamiques et de systèmes qui se développent, tandis que la chimie artificielle offre un moyen de computation idéal et performant dans le milieu incertain et irrégulier de l'ordinateur amorphe. Le *blending*, quant à lui, représente une méthode originale pour la création, l'adaptation et l'organisation de systèmes membranaires.

Les caractéristiques et limites des concepts proposés sont analysées et plusieurs exemples et applications valident les idées introduites à l'aide de simulations. La thèse se conclut par la définition de l'architecture du *Circuit Amorphous Computer* et de l'*Amorphon*, l'élément minimal qui pourrait être à la base des architectures des ordinateurs de demain.

Preface

> "[Individuals who break through by inventing a new paradigm are] almost always [...] either very young or very new to the field whose paradigm they change [...] These are the men who, being little committed by prior practice to the traditional rules of normal science, are particularly likely to see that those rules no longer define a playable game and to conceive another set that can replace them."
>
> — Thomas S. Kuhn
> *The Structure of Scientific Revolutions*, 1962 [227]

When I started with my thesis, I did not only have a lot of crazy ideas, but also quite well knew what I did not want to do. Luckily, there was always plenty of room for visions and "crazy" ideas in the Logic Systems Laboratory. I fully took advantage of this. As things are now, however, this was a rather risky step for my career. A well established researcher might do some moonlighting with such uncommon things, but an unknown PhD student — who desperately needs publications in prestigious scientific journals — should think twice before stepping into such uncertain terrain.

David Patterson's (UC Berkeley) talk entitled "How to Have a Bad Career In Research/Academia"[1] mirrors some of my research in several aspects fairly well. But anyway...I tried something new, something risky, and the contributions made in this thesis are therefore very personal and unconventional. The resulting work is somehow a puzzle of partly unfinished and partly untested concepts that do not (yet) 100% satisfy me. However, my thesis was much too ambitious from the beginning on (which is a very common problem actually, and which is somehow part of this adventure it

[1] `http://www.cs.berkeley.edu/~pattrsn/talks/nontech.html`

seems). Another two or three theses would probably be necessary in order to fully investigate all proposed ideas and concepts.

My years as a PhD student offered highly intense and challenging moments. Now that I'm very close to the end I sometimes have the impression that this thesis started as a dream and ended as a nightmare. I finished being pretty sick of novel computer architectures and crazy ideas... but probably I just need a break and some rest... and I'll be back! In either case, the number of things I learned in the past few years and the experience I was able to acquire are of course extremely precious.

> "I have had dreams and I have had nightmares, but I have conquered my nightmares because of my dreams".
>
> — Jonas Salk

I hope that the few people who'll read this thesis will find something interesting all the same!

Lausanne, December 2003

Christof Teuscher
christof@teuscher.ch
www.teuscher.ch/christof

Acknowledgments

Give us the tools and we will finish the job.

Radio Broadcast
Winston Churchill
9 February, 1974

First and foremost, I'll send my sweetest thanks to my most beloved wife, Ursina, who uncompromisingly and constantly supports me every day and night for twelve years already! Of course I am a little jealous because she beat me and finished her thesis before me ;-). In 1897, Santiago Ramón y Cajal wrote:

> "The selection of the partner! Now we touch on a delicate point. What qualities should grace the young woman chosen by the man of science? This is an extremely serious question because it is undoubtedly true that the moral qualities of the wife are a decisive factor in the success of scientific work. Many are under the shadow of wives for whom they are unsuited — and at times society and even humanity as a whole suffers because of the scholar's wife. So many important projects have been interrupted by the selfishness of the young wife!" [467, p. 102]

Thank god I have wisely chosen my wife! Thanks for everything Ursina!!

Writing a thesis is one thing, but writing both one at the same time borders on sheer madness; every couple who did likewise probably knows what I am talking about. The evenings and weekends we both spend in front of our very faithful five computers are simply uncountable (though finite). Nonetheless we had it more than great — not knowing at the beginning that

so much work could also be fun – and even enjoyed all the conversations about difficult LaTeX and Xfig problems. Writing a thesis is definitely an adventure and a unique experience!

*

I am deeply grateful to Daniel Mange, who supervised this work, and who always made available *any* material, moral and intellectual support for all my numerous undertakings and ideas! From the beginning to the end of my stay in the lab I was always very impressed by his drive, his ideas, his heartiness and empathy, his correctness, his communicational skills, and his capacity to address an audience.

One of the things I most enjoyed in the lab was the almost total liberty (not to be confused with anarchy) we had in research and in buying nice computers every year. And of course I also very much appreciated the unique opportunity and privilege to be alone in an office. This allowed me to fully concentrate on my work without being constantly disturbed.

I hope to have given at least as much to the lab as I was offered and as much as I benefited!

*

Thanks for your invaluable (and often sarcastic) humor, JK, which made many boring lab meeting bearable. I also very much appreciated your numerous direct or indirect advice, which was always very differentiated and without any prejudice. I wish you a very pleasant retirement!

*

Many thanks to Eduardo, who never took life too serious and who was responsible for most of the lab's great barbecues and leisure activities.

*

To "Chico", alias André Badertscher, master of plays on words[2] and heroic guardian of the laboratory, its members and equipment. I learned many exotic words and was much "repudiated" by my colleagues for my many own desperate tries to play with French words. It was always interesting and funny to discuss with you, although I was unable to share most of your rather extreme views. Sorry that we didn't follow your advice to retire in the "Grisons" and to write some further books while enjoying live.

*

[2]Here's a taster: "Qu'est-ce que fait un papa crocodile s'il rencontre une maman crocodile?" ..."Il l'accoste!" :-)

To Ralph, who shared most of my ordeal in completing this thesis and who was a very faithful fellow when we went shooting every spring the mandatory military program with his "tank". He is also one of the few people I know who really do very serious work *only*!

*

To Yann, Fabien, Ralph, and Gianluca for all the amusing (sometimes even moronic) coffee breaks. Thanks also to Yann for the proof-reading of the French abstract.

*

To Auke, with whom I shared the privilege to own a comfortable and relaxing IKEA easy chair in the office. I always very much appreciated your good humor, your kindness, and your professionalism!

*

To Jonas, with whom I had many passionate scientific discussion and shared many points of views.

*

I thank Fabio, Jean-Luc, and Jacques-Olivier for the time we spent together in one office.

*

To Carlos and Andrés, faithful representatives of the once dominant Columbian bastion.

*

To André, who was the most faithful fellow of the legendary "pause de café", which took place every day at 8h45 precisely (Swiss time).

*

Thanks to Alessandro for putting on hundreds of anti-smoke posters on the lab and department's walls and for taking control over *any* machine and computer in his environment. When the elevator once didn't function correctly, we immediately suspected Alessandro who might have found out the elevator's IP address and controlled it via a telnet interface.

*

To Marlyse and Natascha, who did a splendid job with the organization of the Turing Day and IPCAT2003. It was always a pleasure to drop into your office, either to get a difficult administrative problem solved quickly, to get an impossible trip planned, or just for a little chat.

*

To Enrico ("Schätzeli"), who always tried — but never really succeeded — to be nice with me by trying out his newly learned four-letter (actually, the words have a little more letters) German words.

*

I wish to thank wholeheartedly all members of the Logic Systems Laboratory at the Swiss Federal Institute of Technology in Lausanne for making my stay in the lab the most enjoyable and rewarding experience! Thanks for your tolerance regarding all my initiatives, wishes, comments, critics, etc.

*

I am very grateful to my experts, Dario, Marco, Moshe ("The Force. It surrounds us. It enfolds us. It gets us dates on Saturday Nights." – Obi Wan Kenobi, Famous Jedi Knight and Party Animal) and to the president of the jury, Prof. Daniel Thalman, that they have accepted to be on the jury of this thesis. I hope you all the same spent a nice Christmas besides reading this thesis...

*

I am very grateful to Bertrand Mesot for the very fruitful collaboration and the numerous passionate and straightforward philosophical and scientific discussions. You were also the one who was most interested in my own work and with whom I exchanged most books and scientific articles as you had an interest in a lot of topics.

I wish you a very successful career and a lot of pleasure in investigating interesting things or as a Buddhist monk. Do always what you like to do, and you will go very far!

*

Pierre-André, thanks for providing me with almost 1GB of Depeche Mode MP3's, I will miss your unique "SAAAAAAALUUUUUUUUUT!!" when arriving very early in the morning in the lab.

*

I am indebted to Biljana Petreska and Christian Schwarzer for their help, their dedication, and the fruitful collaboration!

*

I thank Barbara Fournier and Grégoire Jotterand from the "Service de Presse et Information" for all their help and commitment during the past few years.

*

To Simon Kramer, who never hesitated to openly criticize my work and ideas and to classify them as "vodoo science" (did you read this [313]?). Thanks, Simon, I always appreciated your honest and most often fairly well-founded criticism!

*

Thanks to the very faithful UNIL sauna community for all the interesting discussions and relaxing evenings! I don't know how Ursina and I would have survived our hard PhD time without that "hot little paradise".

*

I thank the two SUN systems administrators, Ralph and Christof, for their very competent work, their invaluable commitment, and the uncountable days and hours (on evenings and on week-ends too) they had to sacrifice to keep the system running.

*

I am grateful to the Swiss National Science Foundation and the Leenaards Foundation that supported this work.

Contents

CHAPTER 1

Introduction

> ...it was clear that the end was still far, far off, and that the hardest and most complicated part was only just beginning.
>
> *The Lady with the Day*
> Anton Chekhov

1.1 Visions, Paradigms, and Pitfalls

If we believe artificial intelligence (AI) guru Marvin Minsky, who co-founded the MIT Artificial Intelligence Laboratory in 1959 with John McCarthy, "AI has been brain-dead since the 1970s"[1]. In the same talk, Minsky also accused researchers of giving up on the immense challenge of building a fully autonomous, thinking machine. So-called "expert systems", which emulated human expertise within tightly defined subject areas like law and medicine, could match users' queries to relevant diagnoses, papers and abstracts, yet they could not learn concepts that most children know by the time they are three years old.

Marvin Minsky in not alone with his point of view: many a researcher is convinced that the quest for artificial intelligence has turned out to be a failure. Well-known philosophers such as John Searle [362] or Roger Penrose [317] argue that the intrinsic properties of the brain may not be modeled

[1] Mentioned during a recent speech at Boston University, see also: `http://www.wired.com/news/technology/0,1282,58714,00.html`.

by any computer and that the human brain involves other mechanisms. On the other hand, other eminent scientists claim that human-like machine intelligence is only two steps away and that machines will soon become dominant on earth.

Herbert Simon, one of the fathers of AI once said:

> AI can have two purposes. One is to use the power of computers to augment human thinking, just as we use motors to augment human or horse power. Robotics and expert systems are major branches of that. The other is to use a computer's artificial intelligence to understand how humans think. In a humanoid way. If you test your programs not merely by what they can accomplish, but how they accomplish it, they you're really doing cognitive science; you're using AI to understand the human mind."

In traditional AI, it has been tried to specify a problem, i.e., its domain and the conditions, in the most generic and declarative way and to solve it by using a *General Problem Solver (GPS)* [125, 291]. The problem domain is usually modeled as a certain state space with operations leading from one state to another. Today, it seems obvious that the goal of a GPS — i.e., to create a single technique or engine for all problems — cannot be reached. The well-known problems of classical AI are the *Framing Problem* [142] and the *Symbol Grounding Problem* [180] (the domain knowledge has proven to be extremely important in designing efficient problem solving techniques).

However, the goal of modern AI is rather to build special purpose problem solvers that are able to incorporate as early as possible domain-specific knowledge and problem solving methods from other research communities.

In summary: it seems likely that neither the purely pessimistic nor the purely optimistic view of AI will become true. Although current AI does not offer human-like intelligence at all, machines outperform humans in many domains [200]. Machines are usually more powerful in domains where humans have difficulties (e.g., computation, etc.), but they cannot cope with many tasks that humans are able to perform almost unconsciously (e.g., pattern recognition, voice and image recognition, etc.).

Ford and Hayes [143] draw an analogy between artificial intelligence and artificial flight — one of humanity's fondest dreams — and suggest "[...] that the traditional view of the goal of AI — to create a machine that can successfully imitate human behavior — is wrong." In the early 20^{th} century, the American astronomer Simon Newcomb argued passionately against the idea of artificial flight, even after the first aircraft was successfully tested by the Wright brothers in 1903. "The development of aircraft succeeded only when people stopped trying to imitate birds and instead approached the problem in new ways, thinking about airflow and pressure, for example"

[143]. If today's aircrafts lack the bird's elegant precision, they dramatically outperform them in speed: no bird can fly faster than sound and fly at heights where no oxygen is available. Another good example is the wheel: "To appreciate that nature does not necessarily have all the best ideas, we only need point to the wheel. [...] Nature has good reasons to avoid metallic components, for example, but this does not mean that human engineers strive to do so" [25].

It is evident that biological organisms operate on completely different principles from those with which most engineers are familiar. That is why most approaches do not try to faithfully model, duplicate, or copy biological systems but rather try to mimic them and to draw some inspiration from. But many biologically-inspired approaches have been labeled as failures for not having lived up to grandiose promises. "At the heart of this disappointment lies the fact that neither AI nor artificial life (Alife) has produced artefacts that could be confused with a living organism for more than an instant" [51], says Rodney Brooks. Something must be wrong! But what? He proposes different possibilities:

1. we might just be getting a few parameters wrong;
2. we might building models that are below some complexity threshold;
3. perhaps we still lack computing power; or
4. we might be missing something fundamental and currently unimagined in our models of biology.

Another possibility for new and unimaginable discoveries might be some kind of new mathematics. "We may simply not be seeing some fundamental mathematical description of what is going on in living systems and so be leaving it out of our AI and Alife models" [51]. However, none of the mathematical candidate models such as dynamical systems, chaos theories, etc., have so far revealed undiscovered fundamental descriptions.

1.2 Major Problems in Machine Intelligence and Computer Science

Whether you agree with Brooks' proposals (see above) or not, a close look at current intelligent machines (even without being pessimistic) should tell you that there must be something wrong since man-made intelligent machines are pathetic compared to their biological counterparts. The following (certainly incomplete) list enumerates (in no particular order) some of the major and open problems in computer science, and especially machine intelligence encounter today:

- Parallel programming has failed to produce general methods for programming massively parallel systems. Our abilities to program complex systems are simply not keeping up with the desire to solve complex problems.
- Symbolic AI has failed to produce any firm evidence that a symbol-based system can manifest human levels of general intelligence.
- Connectionist models have been unable to faithfully model the nervous systems of even the simplest living things. Although the interconnectivity of the *C. elegans'* 302 neurons in the adult animal is known, it has not been possible to mimic the worm's simple nervous system. One reason is that artificial neural neurons are gross oversimplifications of real neurons.
- Connectionist networks are designed most often by hand and reflect important theoretical claims, experience, and knowledge on the part of the modeler. There are very little general rules on how to design connectionist models. Evolutionary modeling often fails for the lack of computing power.
- There is an enormous gap between cognitive science and cellular neuroscience. Although many ties have been established, it is impossible to describe higher order brain functions on the basis of neuron operations.
- Biological systems are difficult to describe and to model by algorithmic processes.
- Whether the metaphor of the brain or mind as a digital computer is valuable is still an open question. Can cognition be seen as computation?
- We still have a poor understanding of many natural mechanisms, above all at the molecular scale. "The simplest living cell is so complex that supercomputer models may never simulate its behavior perfectly" [160]. The article emphasizes that most attempts to create artificial life or to faithfully model biological systems suffered from a tremendous number of degrees of freedom. The system's parameters could then be tweaked to produce almost any desired behavior. Furthermore, models are often so complicated that they have absolutely no ability to predict anything.
- Most systems do not scale, are not endlessly extensible, and do not allow for open-ended adaptation and evolution. It is, for example, not clear whether the principles of learning and processing in artificial neural systems will work in systems that are several magnitudes more

complex than today. Further, highly complex and extremely large systems might not be able to produce results in real time.

- Emergence and intelligence are ill-defined concepts. There is however little hope that there will be better definitions available soon.
- Most systems are designed (since we do not really know how to do things otherwise).
- It is basically still an open problem how to gradually create more and more complex hierarchical systems.
- How can inanimated matter become alife?
- How can we bridge living and non-living matter?
- How can we build perfect systems out of imperfect components?
- How can complexity been created and how can it been guided during creation?
- How can we create novelty in computer science (not by copy-paste!)
- Are there any "alternative" methods (with regards to today's methods), architectures, organizational principles, etc. which would allow to create better, smarter, and more intelligent systems?
- How can we make further progress in computer science (in general)? Should computation be reinvented?
- Artificial life: Bedau *et al.* [34] state fourteen open problems.

Indeed, machine intelligence is somehow in trouble! Of course, there are no miracles in this field of research, so one should not expect to find an system, an architecture, an approach, etc. that will provide a solution to all this! However, despite all difficulties encountered with intelligent machines, one should nevertheless emphasize that (the core) computer science and science in general have produced staggering complex and large systems such as the internet, mobile phone networks, airplanes, etc. which work to (almost) full satisfaction and which are fairly robust.

1.3 New Architectures and Technologies?

In computer science, the von Neumann architecture (i.e., the "stored program concept") has been for more than half a century (first been expressed in 1945 [443]) largely dominated in many variants and refinements. Even the much-heralded parallel machines of various designs are just collections

of von Neumann machines that possess a communication structure superimposed on the underlying machine. Of course, some alternative approaches were developed, however, they always occupied a marginal place, mainly due to their limited applicability. One might certainly ask if the von Neumann architecture is a paradigm of modern computing science, what is its future? Nobody can anticipate the future, however, there seems to exist a growing need for alternative computational paradigms and several indicators support this observation.

As already seen above with artificial intelligence, the quest for and the interest in novel and unconventional machines and computing paradigms is probably explainable by the somehow disappointing performance of "traditional" machines in some (but of course not all) fields. Furthermore, the post-web services world as well as novel technologies such as nanotechnology have shown a clear need for alternative computing paradigms.

There's a pretty broad spectrum of new and promising possibilities offered, ranging from programmable biomolecules [38], molecular computing [199, 336], programmable matter [417], smart dust [451], to the *utility fog*[2] of the nanotech pioneer J. Storrs Hall, to mention only a few representatives.

Traditional technologies will certainly still capture the industry's interest for at least the decade or so. But ever wondered what will come after them? Amorphous computing is one possible future trend: looking at the billions of smart devices with communication capabilities threatening to flood everyday lives, amorphous computing would probably be a good bet. Amorphous — which means lacking definite form, of no particular type and lacking organization — is a concept that accepts heterogeneity as a way of life, yet allows very different components the ability to interact with one another in their own manner. This allows for many different interpretations for amorphous computing, as we will see later in this thesis.

The 21^{st} century promises to be the century of bio- and nanotechnology. New materials and technologies [144, 196, 256, 474] such as self-assembling systems, organic and molecular electronics, hybrid electronic-biological machines, etc., and the ever-increasing complexity and miniaturization of actual systems [1] will require to fundamentally rethink the way we build computers, the way we organize, train, and program computers, and the way we interact with computers.

Driven by the need to increase performance, scaling down current semiconductors begins to face serious problems. In most current technologies, millions — if not billions — of transistors need to be precisely interconnected and arranged. In molecular electronics, on the other hand, we might make use of random and self-assembled structures, as shown in Figure 1.1.

[2] See `http://nanotech-now.com/utility-fog.htm` for more details.

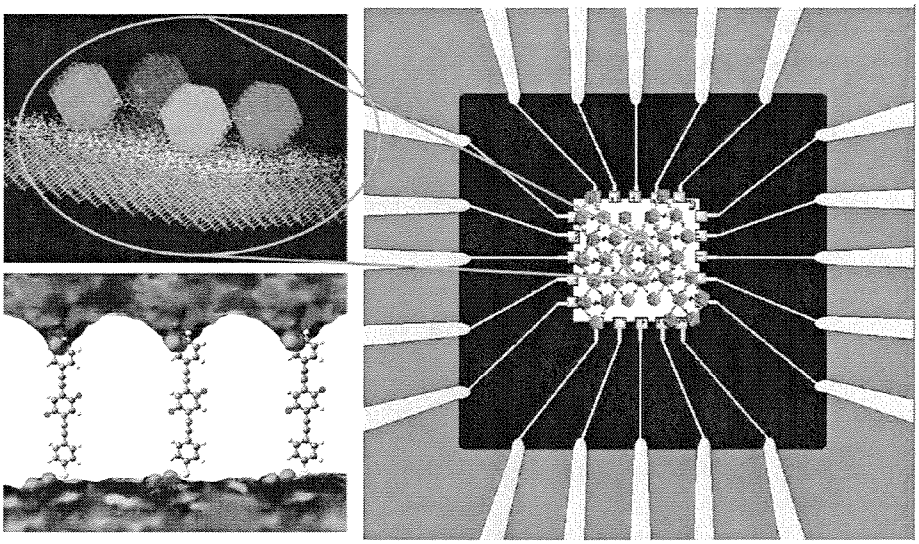

Figure 1.1: Illustration of a molecular random electronics system consisting of self-assembled molecules attached to clusters. The molecular array (right) can be externally interfaced by means of traditional microelectronics technology. Source [366].

Besides nanotech, molecular electronics, and other alternative technologies, there is yet another pretty recent technology which begins to become the more and more relevant: printing techniques to create cheap and flexible sheets of transistors — a process that could radically change the way electronic circuits and for example flat-panel screens are being built[3].

Several labs around the world have been working in recent years to recreate the functions of traditional silicon semiconductor chips with tiny transistors printed on plastic sheets. Using electricity-conducting ink that bonds to a flexible plastic sheet, the goal is to create arrays of bendable semiconductors that work much like their rigid counterparts. Currently, the printable chips are usually not nearly as efficient and reliable as their silicon counterparts, but that is exactly where new computer architectures and organizing principles might be applied (and this is what this thesis is about, isn't it?)

A speculative vision of the future is to replace the huge and expensive "clean room" manufacturing facilities and to create certain semiconductor products in printing facilities that might more resemble a newspaper's printing press, with sheets of transistors scrolling off the high-tech printers.

The trend to new technologies is also clearly emphasized by a recent National Science Foundation sponsored report entitled "Converging Technolo-

[3]The Xerox Palo Alto Research Center (`http://www.parc.com`), for example, is working on such printing techniques.

gies for Improving Human Performance: Nanotechnology, Biotechnology, Information Technology and Cognitive Science" [342]. The report suggest to all researchers to start working in at least one of the four proposed domains (Figure 1.2), which is exactly what we do in the present thesis.

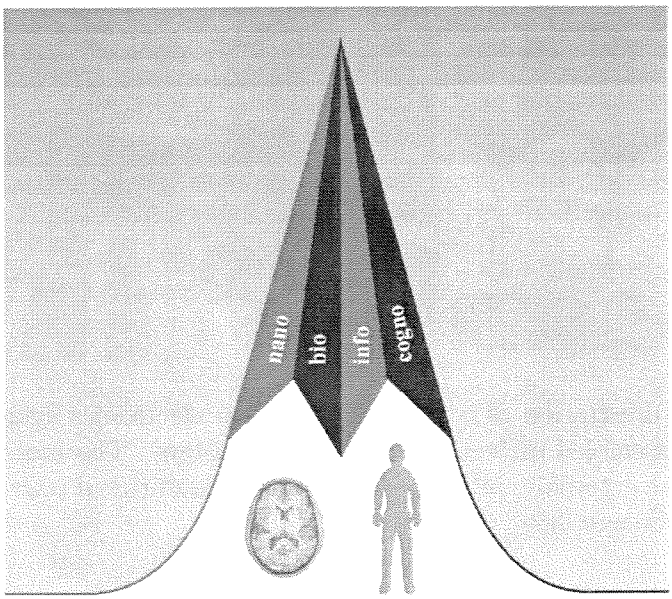

Figure 1.2: Converging technologies for improving human performance: nanotechnology, bio-technology, information technology and cognitive science [342].

1.4 Thesis Statement

1.4.1 Goals and Hypotheses

We have seen above that there are many open problems in computer science and that, in addition, the century promises to be a century of new materials and technologies, such as for example nanotechnology, quantum computers, or new printing technologies for semiconductors.

The present work is nothing else than a humble attempt to seek for further progress in computer science. The goal was not to investigate in an already existing research direction — such as neural networks, etc — but to try something new and "exciting".

The main (and certainly rather visionary) goals of this thesis might be summarized as follows:

- Investigate and pave a possible way to new, unconventional, minimal, massively parallel, imperfect, and specialized hardware architectures.
- Make everything as local as we can.
- Do not impose regularity, i.e., abandon the regularity of interconnections as a random topology might potentially exploit new manufacturing technologies.
- Propose an alternative, irregular computational architecture to the Embryonics (see Section 4.2) and to the POEtic (see Section 4.4) project.
- Abandon the synchronous operation of the system's components.
- Try out a new organizational paradigm inspired by cognitive science.
- Investigate the properties of randomly connected machines and networks in view of new cellular and biologically-inspired computing machines and new manufacturing technologies.
- Propose a possible alternative to the well-established von Neumann paradigm and provide a massive parallel and asynchronous computational model.

In addition, the thesis is based on a set of — more or less precise — hypotheses. We will come back to them throughout the work.

Hypothesis 1.4.1 (Random Structures)
Randomly assembled structures can be more powerful than regular structures (such as cellular automata). ■

Hypothesis 1.4.2 (Future Technologies)
Future technologies such as nanotechnology, new printing technologies for integrated circuits, etc. require radically new concepts and engineering approaches. ■

Hypothesis 1.4.3 (Perfect Systems)
Building perfect systems out of imperfect components is likely to become the more and more important in the future, mainly due to the ever growing number of components involved. ■

Of course, perfect systems are not always required: in domains related to soft-computing [407] (e.g., neural networks, fuzzy logic, etc.), it is very often sufficient to "build imperfect systems out of imperfect components".

These three hypotheses are nicely summarized by the following quotation, which might be considered as a sort of "spiritual mantra" and driving force for this thesis, although the goal was *not* to provide any directly applicable method to program or build molecular electronics:

> "Molecular electronics can be developed if we are able to program a random arrangement of molecules or a field-programmable molecular random array" [366].

Finally, I'd like to draw the reader's attention on some warnings and important points to remember while reading this thesis:

- The present work addresses fundamental questions in complex systems, computer science, and unconventional computing paradigms and must therefore for be considered as fundamental research which provides new and mainly theoretical concepts only.
- The goal is to investigate and propose new paradigms and concepts and *not* to find new applications or to outperform existing systems.
- The choices made and concepts proposed were more influenced by practical and implementational than by theoretical issues. I always tried to keep in mind a possible implementation.
- Real-world killer applications for the concepts proposed are probably several years ahead.
- I do *not* provide any methodology to program membrane systems and artificial chemistries. This remains an open (and at the same time fundamental) problem.
- Simulations and examples — if existent — are restricted to ridiculously simple and small toy applications to prove the main concepts. The principal reason for this are the computational limitations which cannot be overcome today. It is virtually impossible to simulate large amorphous computers, or even more general, to simulate a large number of parallel working components.
- Apart from a very small prototype built at MIT, no amorphous computer has ever been built. The work done in this thesis was therefore from the beginning on condemned to remain in the realm of simulations.
- It was my intention to keep the simulator programmed in MATLAB as simple as possible and not to offer graphical user interfaces. The goal was to proof the concepts, not to offer a "sellable" tool.

- As several topics of this thesis are somehow less known in the biologically-inspired computers science society (especially the blending part), I made a considerable effort to provide fairly complete introductions and examples, which of course bloated the work.

1.4.2 Approach

In this thesis, a rather unconventional combination of basically three different domains of research is presented:

1. amorphous computing,
2. membrane computing, and
3. blending.

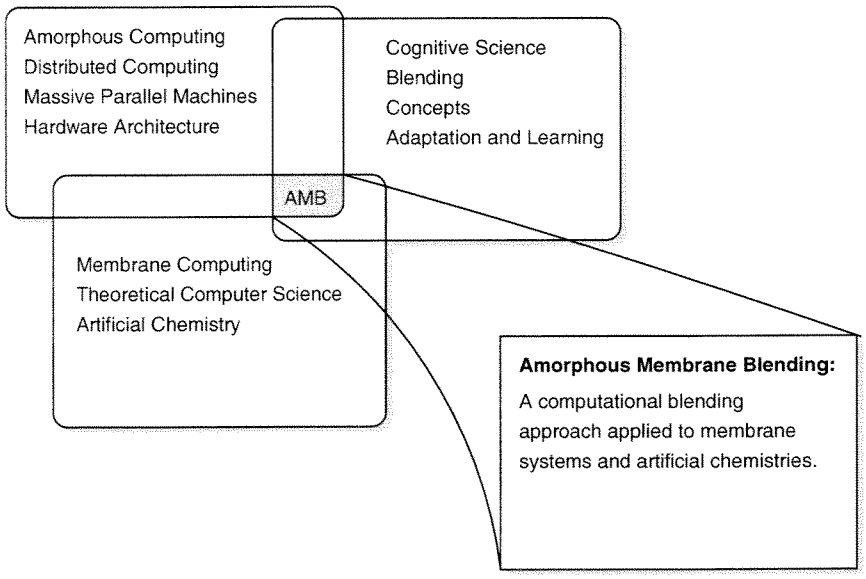

Figure 1.3: An overview on the various domains involved in amorphous membrane blending.

Figure 1.3 illustrates the principal domains involved in this approach, which shall me called *Amorphous Membrane Blending* (AMB).

From a bird's eye view the amorphous membrane blending landscape can be decomposed into four principal hierarchical levels as depicted in Figure 1.4:

1. reactor level,
2. amorphon level,

3. cellular level, and
4. supercellular levels.

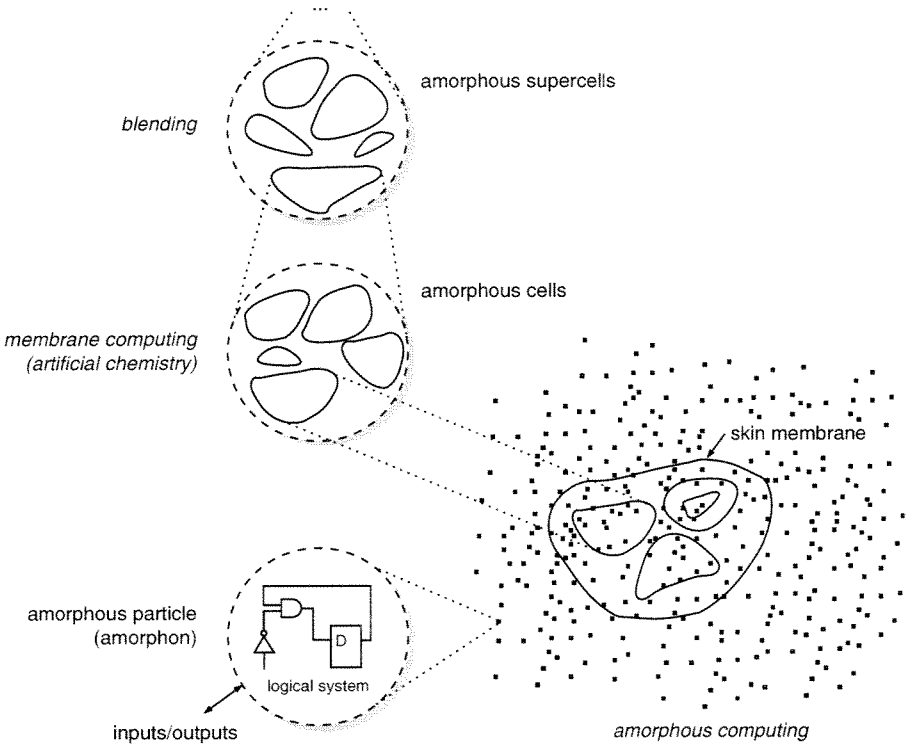

Figure 1.4: The amorphous membrane blending landscape can be composed into four principal hierarchical levels, although cells can form hierarchies of almost any complexity.

Throughout this thesis, we will come across all of this levels.

I completely agree that the approach might seem rather weird at a first glance (and maybe even after a profound investigation), but it was neither the goal to provide a biologically-plausible model nor to make use of "conventional" techniques and methods. Please keep that in mind while reading this thesis (and please also carefully read the warnings of Section 1.4.1)!

Finally, I'd like to provide some precursory answers to the probably most central and burning questions. I hope that this answers will help to facilitate the further reading.

1. *Question*: Why cells and membranes? *Answer*: Because they allow to build hierarchical structures and to divide complexity. They are also a mean to divide a task into several sub-problems and to provide a mode of parallelism.

2. *Question*: Why an artificial chemistry? *Answer*: Because it is an ideal mean to compute in uncertain environments and because artificial chemistries have been identified as potentially very promising for the perpetual creation of novelty (see for example [166]).

3. *Question*: Why an amorphous computer? *Answer*: Because it is a promising paradigm for future computational substrates and technologies such as molecular electronics or printable digital circuits.

4. *Question*: Why using irregular and imperfect cellular machines? *Answer*: Because its seems that they will play a crucial role for new computational substrates and environments. It is also often hypothesized that nanotechnology will need approaches which are able to compute on the basis of an irregular and imperfect medium.

5. *Question*: Why blending? *Answer*: Because it appeared to be a potentially interesting way to create new cells from two existing cells.

1.4.3 Contributions

The following principal contributions were made in this thesis:

- An overview and review of some promising POEtic machines, architectures (Section 2.8) and other unconventional models of computation. This overview was principally made in order to delineate promising new directions and to eliminate possible dead-ends.

- The BioWall hardware architecture was proposed (Section 4.3). The same work was also at the origin of a patent proposal [247] and of various publications [389, 390, 404, 406, 412]

- Implementation of a first hardware reconfigurable implementation of P systems for classical and regular hardware architectures (Section 4.8). The implementation supports priorities and allows to dissolve and create membranes. Experiments and performance results are also provided.

- The synchronization task for two-dimensional cellular automata (Section 3.5) and for random boolean networks (Section 3.7) has been investigated and implemented. A co-evolving realization on the BioWall has been implemented (Section 4.3.4) as well as a Java program that allowed to verify the co-evolving genetic algorithm.

- Realization of a MATLAB random boolean network toolbox (Section 3.8) that allowed to explore various properties of random networks.

- Investigating the dynamics and topological evolution of random boolean networks with different updating modes (Section 3.9) according to the classification of Gershenson [158].
- Investigating critical values in asynchronous random boolean networks (Section 3.10). No critical values could be found.
- Development of the cellular blending (C-Blending) framework (Section 5.6) which allows to create new membrane systems inspired by Fauconnier and Turner's blending.
- Propose atP membrane system as an example and candidate model (Section 6.4)
- Propose aP membrane systems (Section 6.3), a special class of membrane systems with several features which facilitate a distributed hardware implementation. This membrane system not only allows to rewrite molecules but also reactions, which might be considered a molecules. Section 6.3 also includes examples of the implementation of (redundant) state-machines, boolean circuits, and robust oscillators using aP chemistries.
- Propose aB membrane system which unifies the computational C-Blending approach with aP membrane systems.
- Propose the Circuit Amorphous Computer, an modified amorphous computer (Section 7.2) used as a computational substrate.
- Propose the abstract *Amorphon* architecture (Section 7.3).
- Creating hierarchical membrane structures on an amorphous computing substrate (Section 7.4). The approach is based on a chemical metaphor with a distributed reaction network.
- Implementing the MATLAB Amorphous Membrane Blending toolbox. Many examples of how to used the functions are directly included in the text. A function reference is provided in the appendix.

1.5 Thesis Organization

The thesis is organized as follows:

Chapter 2 provides a rather general overview on computation (from the point of view of a computer scientist) and on natural computation (also called biologically-inspired computation or soft-computing). Special emphasis is put on the so-called POEtic machines, i.e., machines which amalgamate evolution, learning, and developmental mechanisms (see Sections 2.7 and 2.8).

This chapter also introduces two of the main ingredients of this work, namely amorphous computers and membrane systems. A brief introduction on artificial chemistries is also provided as they are closely related to membrane systems.

Chapter 3 deals with regular and irregular computational structures. Cellular automata are probably the most popular representatives of regular and homogeneous structures, whereas random boolean networks present an irregular model might might also used nondeterministic node updating schemes. In order to investigate the properties of random boolean networks, a MATLAB toolbox has been implemented. The chapter also presents results of a self-organized topological evolution algorithm which has been investigated by using different node updating schemes. Finally, an analysis of critical values in asynchronous random boolean networks is presented.

Chapter 4 focuses on biologically-inspired hardware. The main parts are dedicated to the Embryonics project (Section 4.2), the large-scale BioWall cellular automata machine (Section 4.3), and to a first reconfigurable implementation of membrane systems on Field Programmable Gate Arrays (FPGAs) (Section 4.8). The implementation also allows to dissolve and create membranes. The chapter contains also briefly mentions some other interesting hardware architectures inspired by biology.

Chapter 5 first introduces concepts, mental representations, and the basics of conceptual integration (or blending). Section 5.5 provides a field review on the few computational blending approaches that exist.

In the last section, a new and unique computational blending approach — entitled C-Blending— is presented which deals with membranes and artificial chemistries instead of mental spaces and concepts.

In Chapter 6, several special membrane systems are introduced. atP, aP, and aB membrane systems were proposed in view of an implementation an an amorphous computer. The implementation of boolean circuits as well as state machines and oscillators are analyzed for aP membrane systems.

Section 6.4 presents aB membrane systems which unify C-Blending and aP systems.

The final chapter basically deals with implementational issues at the level if the amorphous computer and its particles. First, a modified amorphous computer model, the Circuit Amorphous Computer, is proposed. The Circuit Amorphous Computer has a modified communication model which seems to be more suitable for novel integrated circuits and printed sheets of transistors. In the second part, the amorphous computer's particle, the amorphons, are presented and their principal operations described.

CHAPTER 2

Computation, Natural Computation, and Unconventional Models of Computation: An Overview

> The problem with computers is that there is not enough Africa inside.
>
> Brian Eno

2.1 Introduction

Natural computation[1] is the study of computational systems that use ideas and get inspiration from natural systems, including biological, ecological and physical systems (Figure 2.1). It is an emerging interdisciplinary area in which a range of techniques and methods are studied for dealing with large, complex, and dynamical problems. Typical topics include the following, but are not limited to:

- evolutionary computation;
- evolvable hardware;
- quantum computation;
- artificial life (AL);

[1] The terms *biologically-inspired computation* (see for example [248, 373]) or *soft-computing* [407] are also frequently used.

- self-organizing systems;
- emergent behaviors;
- machine learning and perception;
- robotics;
- neural networks;
- applications to real world problems.

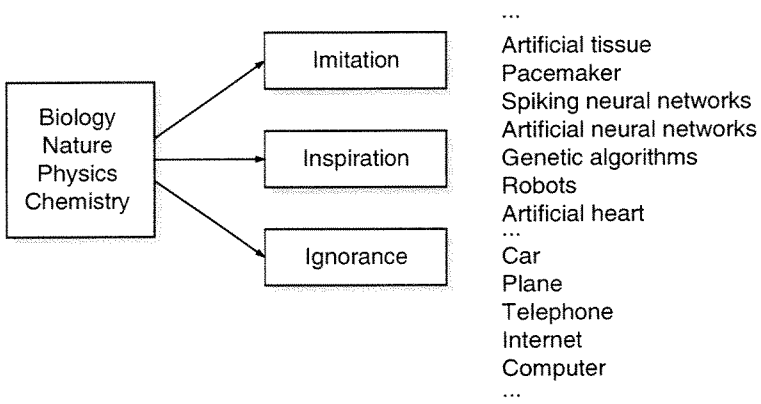

Figure 2.1: Biological inspiration or not: engineering borrows many ideas from Nature.

A showcase example I personally like is the development of tomorrow's billion-transistor circuits and of electronic components of atomic dimensions which will confront computer scientists and engineers with a challenge that John von Neumann already tried to face in the fifties: *building perfect systems out of imperfect components.* It is rather likely that this challenge will have to be taken up by "alternative" approaches, such as by looking at Nature and by imitating some of its most interesting traits to provide architectural and organizational principles that will allow to build perfect systems from billions of partially imperfect components. This point of view is of course not shared by the entire research community, nevertheless is a good example where natural computation might come in.

Natural computation is a term that is pretty often also misused. For example, it has nothing (or at least very little) to do with *bioinformatics.* The very definition of bioinformatics is, however, still the matter of some debate too. Roughly, bioinformatics describes any use of computers to handle biological information. Although some interpret it narrowly as the information science techniques needed to support genome analysis, many have begun to use it synonymously with "computational molecular biology" or even all of "computational biology".

One of the main (and old) driving questions of science is certainly still the quest for understanding how life has originated on the planet Earth (see for example Nobel laureate Erwin Schrödinger's "What is Life?" [359]). A lot of questions remain open and many theories are very controversial (see for example [430]). Directly related to this question is quite naturally the question whether life might be created *artificially*, e.g, *in silico* or *in vitro*, and whether it will one day be possible to create machines with human like *artificial intelligence (AI)*.

In defining living systems, there are basically two complementary schools of thought that have concentrated on two fundamentally different properties of life: the primacy of *self-replication* [121] and the primacy of *autopoiesis* [262, 434, 435].

Life itself might be defined as a chemical system capable of self-reproduction and of evolving. It is generally assumed that the first life originated from the reaction of reduced carbon-based organic matter on the primitive Earth. Primitive carbon was available as gaseous compounds, either oxidized (carbon dioxide, carbon monoxide) or reduced (methane). More complex organic molecules might have been formed by the action of UV light, shock-waves and electric discharges in a primitive atmosphere.

The (biological) origin of life requires basically an assortment of molecules that do some essential processing: the must be able to catalyze reactions that lead—directly or indirectly—to the production of more molecules of the catalyst itself. Such a system is called *autocatalytic* and usually has some of the properties we think of as characteristics of living matter: a tendency to reproduce itself and a selection of the interacting molecules. As Stuart Kauffman states: "[...] the secret of life, the wellspring of reproduction, is not to be found in the beauty of Watson-Crick paring, but in the achievement of collective catalytic closure" [213].

Stanley Miller's classic 1953 experiment [269] (see also [270,271]) demonstrated the mechanisms by which inorganic elements could combine to form the precursors of organic chemicals. He exposed (see Figure 2.2) a mixture of methane, ammonia, hydrogen and water to electric discharges (lightning of the primitive earth atmosphere) to mimic the action of lightening on a primitive atmosphere. He then obtained four of the twenty amino acids utilized in life today, via the intermediary formation of hydrogen cyanide and aldehydes. In the laboratory, hydrogen cyanide and formaldehyde have been shown to lead to many of the building blocks of the biopolymers, such as amino acids and nucleic acid bases.

It is of course not astonishing, that most scientific "origin of life" theories are rejected by the creationists, but this topic would fill another thesis...

Biology is the scientific study of life on earth based on carbon-chain chemistry. However, there is nothing in its charter that restricts biology to carbon-based life — it is simply that this is the only kind of life that has been available to study.

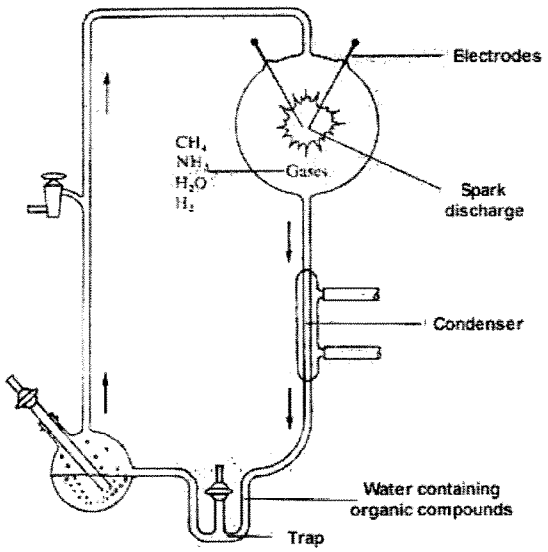

Figure 2.2: The Miller "origin of life" apparatus.

Artificial life (Alife, a-life, literally "life made by Man rather than by Nature"), on the other hand, is the study of synthetic systems which behave like natural living systems in some way. Artificial life complements the traditional biological sciences concerned with the analysis of living organisms by attempting to create lifelike behaviors within computers and other artificial media. Artificial life can contribute to theoretical biology by modeling forms of life other than those which exist in nature. It has applications in environmental and financial modeling, network communications, optimization techniques, etc. Typically, artificial life is considered as a *bottom-up* approach to synthesize primitive artificial life forms.

The reader interested in artificial life is referred to the work of Langton [233, 235, 236], Adami [5], Fontana [139–141], Rennard [339] (a nice book in French), and many others.

2.2 Machines and Computability

Most books on theoretical computer science and astonishingly also on natural computation start with a defining of what "computation" is and what a Turing machine can do and cannot to. Whereas the concept of "computability" is undoubtedly central to theoretical computer science, its importance might be questioned with regards to biologically-inspired computing machines and Nature as they exactly do the opposite: they do *not* "compute" (if one can really talk about computation) universally but produced highly specialized units instead which are all but universal (see also [472]). My

personal conclusion is that universal computation in the domain of natural computation is a totally irrelevant concept. Likewise, Turing machines are mostly irrelevant to artificial intelligence (see also [380]). Nevertheless, a overview shall be given here, especially also on an emerging field termed *hypercomputation*, i.e., computation beyond the Turing limit.

Turing proposed the Turing machine in 1936 [422] (see also Section 2.2.1) as a model of a mathematician that solves an algorithmic problem using paper and pencil. Along with others, including Church, Post, Gödel, Kleene, etc., Turing laid the fundamentals for modern computing science [409]. The Church-Turing thesis (CT thesis) states that a universal Turing machine can carry out *any* effective computation, i.e., it can simulate any other machine capable of performing a well-defined computational procedure. In this context, "effective" is a synonym for "mechanical". There exist many different formulations of the Church-Turing thesis and it has often been misinterpreted. Copeland and Sylvan [87] provide an overview of several typical statements. Trakhtenbrot [420] provides a comparison of the Church and the Turing approaches. Interested readers might also take a look at Hasslacher's paper "Beyond the Turing Machine" [183] and at Galton's paper about the nature of the Church-Turing thesis [152]. He argues that the study of complex, nonlinear, and self-organizing systems might lead to the construction of new kinds of computational models which cannot efficiently be simulated by the traditional concept of a Turing machine.

One of the most accessible formulations of Turing's thesis is probably the following:

> It is found in practice that LCMs can do anything that could be described as 'rule of thumb' or 'purely mechanical' [426, p. 7].

The term "LCM" stands for logical computing machine (see also Section 2.2.1). Another definition is the following:

> A function computable in any reasonable computational model is computable by a Turing machine [115].

Du and Ku naturally ask what a "reasonable computational model" is and provide the following, again rather intuitively defined points:

- The computation of a function is given by a set of finite instructions.
- Each instruction can be carried out in this model in a finite number of steps, or in a finite amount of time.
- Each instruction can be carried out in this model in a deterministic manner.

It is important to be aware, that, the Church-Turing thesis is not well defined mathematically, it cannot be proved and we can only collect mathematical proofs as evidence to support it.

Modern versions of the Church-Turing thesis often state that no realizable computing device could be more powerful—aside from the speedup—than a universal Turing machine. Even a strong version of the thesis—stating that no realizable physical device can be more powerful than a Turing machine—has not been refuted. Indeed, nobody has found so far a physical computing device that is computationally more powerful than a Turing machine. Even *quantum computing* has been proved by Deutsch in 1985 to be UTM-equivalent only:

> "Every finitely realizable physical system can be perfectly simulated by a universal model computing machine operating by finite means" [107, p. 99].

The Church-Turing thesis has even been (ab)used to formally approach the fuzzy notion of intelligence:

> "What is human computable is computable by a universal Turing machine" [58].

Such definitions are of course highly speculative and most often lack any serious scientific reasoning. The problem already begins with the definition of what the brain "computes". Michael Conrad, who unfortunately passed away much too early, was one of the pioneers in the domain of brain-machine disanalogy and in investigating biological information processing(see for example [76,77,473]). Conrad believed — and was probably right — that there are fundamental lessons to be learned from the structure and behavior of biological brains that we are far from understanding or have implemented in our computers. For example, his assertion that programmable computers are intrinsically incapable of the brain's efficient and adaptive behavior has unfortunately not received much examination.

More shall be talked about this issues later, but let us first come back to purely theoretical, but not less important, issues in computer science.

The following two (classical) definitions regarding computability shall be briefly mentioned before we will enter the field of Turing machines:

Definition 2.2.1 (Unsolvable Problems)
A formally stated problem is unsolvable *if no Turing machine exists to compute the solution. A formally stated problem is provably unsolvable if it can be proved no Turing machine exists to compute the solution.* ■

Definition 2.2.2 (Undecidable Problems)
A formally stated problem is undecidable *if no total recursive function and thus, no Turing machine that always halts, can be constructed to decide the problem, usually true or false.* ■

The most famous unsolvable problem is probably the *halting problem* is a very strong, provably correct, statement that no one will ever be able to write a computer program or design a Turing machine that can determine if a arbitrary program will halt (stop, exit) for a given input. Of course, this does not mean that some programs or some Turing machines cannot be analyzed to determine that they, for example, always halt. The halting problem says that *no* computer program or Turing machine can determine if *all* computer programs or Turing machines will halt or not halt on *all* inputs.

The two above-stated definitions are closely related to *Gödel's Incompleteness Theorems* (simplified versions):

Theorem 2.2.1 (Gödel's First Incompleteness Theorem)
In any consistent formal system of mathematics sufficiently strong to allow one to do basic arithmetic, one can construct a statement about natural numbers that can be neither proven nor disproven within that system. ■

Theorem 2.2.2 (Gödel's Second Incompleteness Theorem)
Any sufficiently strong consistent system cannot prove its own consistency. ■

In other words, Gödel's first incompleteness theorem says that any sufficiently strong formal system of mathematics is either inconsistent or incomplete. Note that a *complete formal system* is a formal system where all true theorems can be proved whereas an *inconsistent formal system* is a formal system where at least one false statement can be proved within the formal system.

Finally, let's briefly talk about formal language theory: a *formal language* is defined as set of finite strings over an alphabet of finite symbols such as for example a and b. A typical string over this alphabet would be *abbaab*, and a typical language over that alphabet containing that string would be the set of all strings which contain the same number of a's as b's. Note that while the alphabet is a finite set and every string has finite length, a language may very well have infinitely many member strings.

A formal language can be specified in a variety of ways. The most common ways are strings produced by some formal grammar , string produced by a regular expression (as known in UNIX/Linux systems), or strings accepted by some kind of automaton (finite, Turing, etc.).

A decision problem can be posed as given a language is a given string in the language. Basically this is the mathematical problem of given a set is a particular element in the set.

The classes of formal languages can be described by the *Chomsky hierarchy*, which has been described by Noam Chomsky in 1956 [66].

The Chomsky hierarchy consists of the following levels:

- **Type-0 grammars** (unrestricted grammars) include all formal grammars. They generate exactly all languages that can be recognized by a Turing machine.
- **Type-1 grammars** (context-sensitive grammars) generate the context-sensitive languages. These languages are exactly all languages that can be recognized by a non-deterministic Turing machine whose tape is bounded by a constant times the length of the input.
- **Type-2 grammars** (context-free grammars) generate the context-free languages. These languages are exactly all languages that can be recognized by a non-deterministic pushdown automaton. Context free languages are the theoretical basis for the syntax of most programming languages.
- **Type-3 grammars** (regular grammars) generate the regular languages. These languages are exactly all languages that can be decided by a finite state automaton. Additionally, this family of formal languages can be obtained by regular expressions. Regular languages are commonly used to define search patterns and the lexical structure of programming languages.

2.2.1 Turing Machines

In his seminal 1936 paper [422] (reprinted in [92] and [204]), Turing described a certain type of machine called *logical computing machine (LCM)*. Today, this machine is more commonly known as the *Turing machine*. A Turing machine—an abstract computing device—is a finite-state machine associated with an external storage or memory medium (see Figure 2.3). For example, this storage medium can be a linear tape that is regarded as infinite in both directions. The machine is coupled to the tape through a head, which is situated, at each moment, on some square of the tape. The head then can read and write information from and to the tape. Furthermore, it can move to an adjacent square. A Turing machine computes via a sequence of discrete steps and its behavior is always completely deterministic. The tape of a Turing machine is often considered as infinite in both (or one) direction. However, this view of idealization is physically completely unrealistic and only holds as an abstract model. A better approach is to

view the tape as finite, but indefinitely extendible, i.e., whenever an additional square is needed, one can be attached to either end of the tape. The program of a Turing machine is normally given in the form of a discrete state transition diagram.

Many Turing machine variations have been proposed (e.g., multi-tape Turing machines, probabilistic Turing machines, multidimensional Turing machines, etc.), however, any variation can always be simulated (often with a loss of time) by a one-tape Turing machine. Turing machines are a simple and yet powerful enough computational model.

For further reading about Turing machines, interested readers are referred to [92, 93, 134, 192, 272, 422].

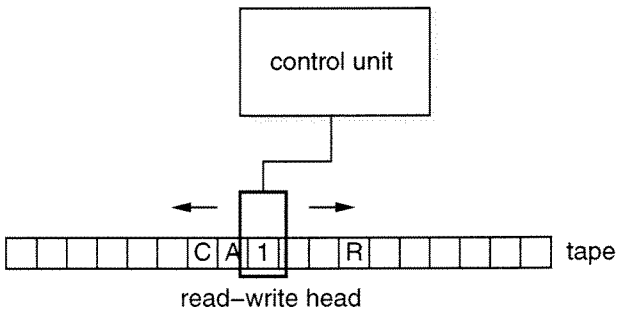

Figure 2.3: Architecture of a Turing machine: finite state control unit, tape, and read-write head.

It is possible to program a Turing machine to accept the description of the program and input data of any other Turing machine computation, and to simulate that computation. Such a machine is called *universal logical computing machine* (ULCM) or *universal Turing machine* (UTM). The universal machine thus simply carries out the operations of the machine whose description was given on the tape. Today, one would rather say that the machines execute a *program*. The necessary components of the machine U are (1) a finite-state machine (the program of U) that controls the mobile head operating on the tape; (2) the data on the tape that describes the specialized Turing machine T to be simulated (the data of T, and (3) the program of T.

However, what is the interest of a UTM? Turing wrote:

> "The importance of the universal machine is clear. We do not need to have an infinity of different machines doing different jobs" [426, p. 7].

Figure 2.4 shows the organization of a UTM's tape as given in [272]. A first semi-infinite region contains the data of *T's* tape and a marker M indicating where *T's* head is currently located. The second region contains

the current internal state Q and the current input symbol S of T. Finally, the third region is used to record the description of T, i.e., the three functions Q^+, S^+, and D^+ for each combination of Q and S. For an in-depth description of the UTM see [272, p. 132–144]

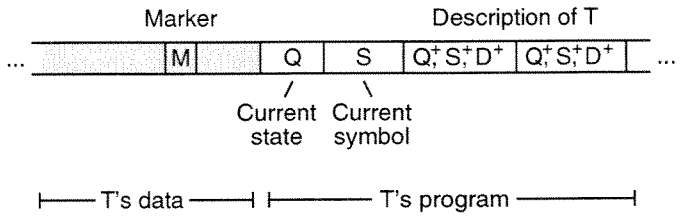

Figure 2.4: The tape of a universal Turing machine U containing the description of a specialized machine T.

Probably the most important thing to say about Turing machines is that—as an abstract model—they are functionally as powerful as any computer. This statement—misinterpreted many times—is generally known as the *Church-Turing thesis*. So far, the Church-Turing thesis has not been refuted and will most likely remain valid. For more details about the thesis and hypercomputers—machines that compute beyond Turing machines—see also Section 2.2.2. For further reading, the interested reader is also referred to Davis' book about universal computers [93].

It is important to be aware that there exists a provable equivalence between the following main formal models of computation:

- Turing machines,
- Lambda Calculus,
- Post Formal Systems,
- Partial Recursive Functions,
- Unrestricted Grammars,
- Recursively Enumerable Languages, and
- intuitively what is computable by a computer program written in any reasonable programming language.

Of course, *any* modern computer and all general-purpose programming languages also compute universally and can simulate/emulate/implement any of the above-mentioned computational models. One also says that a machines is *Turing-complete* when it has computational power equivalent to a universal Turing machine.

To prove that a machine can compute universally, it is basically only necessary to prove that it can compute a NAND function and that they can be assembled to a more complex machine. NAND functions form a logical basis [447] and it is therefore possible to realize *any* logical function on the basis of NAND gates. Other logical basis exist: John von Neumann [444] showed that through a redundant coding of the inputs (each variable is transmitted through two lines) AND and OR units alone can constitute a logical basis as well.

In the context of the famous *Game of Life*, a Berlekamp *et al.* stated: "It is possible to construct AND, OR, and NOT gates using the *Game of Life*." [...] "From here on it's just an engineering problem to construct an arbitrarily large finite (and very slow!) computer" [39, p. 841]. This challenge has recently been met: Paul Rendall [337] describes in detail how to build a Turing machine from the patterns of the *Game of Life* (see also [338]).

A machine that can only compute one single NAND function can of course not be considered as a universal computer. The definition above is somehow a weak one as it assumes that the NAND gates are available in a sufficient number and that they can be assembled to a more complex machine, e.g., a Turing machine.

I'd like to further emphasize some points related to building universal machines from NAND gates (or other gates which form a logical basis, such as AND or NOT gates). First, one has to make a clear distinction between the conceptual and the engineering point of view. From the conceptual point of view, NAND gates are definitely sufficient to realize any logical system, including sequential logical systems. As Figure 2.5 illustrates, it is easily possible to realize an *positive-edge-triggered D flip-flop* based on NAND gates only. A positive-edge-triggered D flip-flop samples its D input and changes its Q output only at the rising edge of the controlling clock (CLK) signal and can therefore be considered as a very simple memory element. Note that the idealized (i.e., theoretical) NAND gates do *not* possess (or have to possess) any internal delay. There is one small detail to consider: the sequential circuit requires an external clock signal for a correct operation. However, this is an irrelevant issue from a theoretical point of view since a Turing machine *also* requires an external clock signal to function correctly!

From an engineering point of view, there are many other issues to consider of course. For example, it must be ensured that all necessary connections between the components can be established, that the clock signal can be distributed, that the timing constraints of the components are respected, that each component receives enough electrical power, etc. But all this is "just an engineering problem", nothing more and nothing less.

In summary, being able to compute a NAND function is a necessary and sufficient condition for building a universal computer from the theoretical

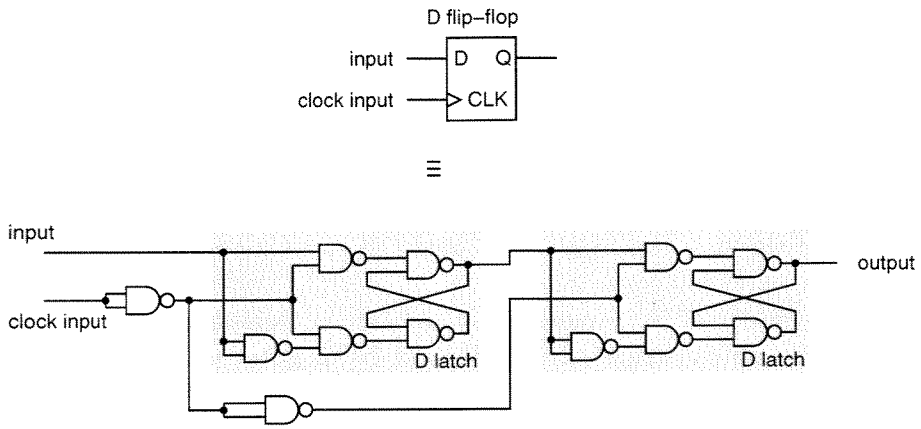

Figure 2.5: An positive-edge-triggered D flip-flop built up from NAND gates only. The circuit requires a clock signal.

point of view, but only a necessary condition from the practical point of view.

2.2.2 Computation Beyond Turing Machines?

The question might be asked whether a universal Turing machine really captures the essence of any and all forms of computing? Or, are there *hypercomputers*, also called *super-Turing machines*, capable of going beyond the Turing limit? Hypercomputation is a pretty controversial field of research concerned with the study of (theoretical and practical) computation beyond the computational capabilities of the Turing machine.

Turing himself presented a model of an abstract hypercomputer—the *O-machine*—in his 1938 doctoral thesis supervised by Church (published in 1939 as a paper [423]). An O-machine is a universal Turing machine augmented by an *oracle* or "black box". The oracle performs a computation which a universal Turing machine operating in finite time cannot compute [86]. Turing used his oracle to describe an abstract mathematical and uncomputable operation. He gave no indication on how such an oracle might be implemented but wrote that:

> "We shall not go any further into the nature of this oracle apart from saying that it cannot be a machine".

Thus, Turing was aware that his O-machines could not be implemented physically! It is important to note that any speculation on building physical hypercomputers—computers that compute the uncomputable—has *no* basis in Turing's writings. This should, however, not prevent researchers from further investigations on possible physical super-Turing machines since neither Church nor Turing had demonstrated the impossibility of hypercomputers.

As already stated above, so far, nobody has found a physical computing device that is computationally more powerful than a Turing machine. There exist, however, many *Gedankenexperiments* in idealized and hypothetical universes that allow the construction of non-physical hypercomputers. One of the best known hypercomputer models has been presented by Hava Siegelmann. The model is based on analog recurrent neural networks (ARNN) [367–369]. An analog recurrent neural network is a finite network of neurons and connections wherein the synaptic weight associated with each connection is a real (analog) value. Siegelmann showed that analog recurrent neural networks are more powerful than the Turing-machine model in that they can perform computations provably uncomputable by a universal Turing machine. When the real synaptic weights are replaced by rational numbers the network's computational power is reduced to that of a Turing machine only. However, as Davis recently showed [94], Siegelman's approach is mostly nonsense and based on wrong assumptions. It is not astonishing that a machine can compute all languages when all languages (uncomputable real numbers) are fed in!

The *accelerating universal Turing machine (AUTM)* [82, 83, 393] is a hypercomputer that executes the program on its tape at an accelerating rate. The first operation needs one time unit to complete, the second 0.5 time units, the third 0.25 time units, etc. The accelerating universal Turing machine requires a maximum of two time units to execute any possible program. Thus, even a program that does not terminate would be finished in two units only. An accelerating universal Turing machine can compute the infinite in finite time! That's probably the moment one must admit that *Gedankenexperimente* have limited validity only.

Definition 2.2.3 (Execution time of an AUTM)

$$1 + \frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^n} + \cdots = 2 \tag{2.1}$$

■

Mike Stannett presented another hypercomputer model—the analogue X-machine—and showed that it can solve the classical *halting problem* that no Turing machine is able to solve [386]. The approach is based on the fact that true analogue systems are computationally more powerful than discrete systems. Just as no universal Turing machine could ever be built since we could not provide an infinite tape, it is very likely that Stannett's analogue machine could never be built.

The practicality of physical hypercomputation has, in fact, been questioned by several researchers. Most hypercomputer models involve analog computation with infinite precision or try to compute the infinite in finite

time. Given the lessons of quantum mechanics, it seems Nature would not tolerate our building infinitely precise machines. Moreover, the effect of noise on analog computation presents yet another obstacle to the implementation of hypermachines. As for quantum computation—wherein physical phenomena at the quantum level are directly employed to build more powerful computers (at least in theory)—it is too early to tell whether this domain holds any promise where hypercomputation is concerned. This is not to say that quantum computers cannot compute much *faster* than classical ones.

The first critical essay on hypercomputation — an absolute must for any "believer" — comes from the distinguished logician Martin Davis: "The Myth of Hypercomputation" [94]. Davis analyzes typical hypercomputationalist claims and concludes that when viewed critically, they amount to little more than the obvious comment that if non-computable inputs are permitted, then non-computable outputs are attainable.

Assume, nonetheless, you have managed to overcome these implementation obstacles and (pseudo-)implement a hypercomputer; to what use could you now put it? Perhaps you could build a "brain": one of the major application domains of hypercomputation being envisaged is that of machine intelligence. This raises the question of whether the human brain itself is a super-Turing machine, an issue over which opinions diverge widely. Most computational brain models presented to date are less powerful than a universal Turing machine. On the other hand, the modest success of classical artificial intelligence has urged researchers such as Mike Stannett to speculate that:

> "[i]f biological systems really do implement analogue or quantum computation, or perhaps some mixture of the two, it is highly likely that they are provably more powerful computationally than Turing machines" [386].

This statement implies that truly intelligent behavior cannot be implemented on standard digital machines, an opinion shared by Roger Penrose [317, 318] who believes that mechanical intelligence is impossible since purely physical processes are uncomputable. If it turns out that the universe is in fact continuous, then it would seem in fact very unlikely that computational processes tell the full story of how nature works. The interested reader is also referred to Copeland's paper on "Turing's O-machines, Searle, Penrose and the brain" [84].

Turing himself believed that the brain should be considered as a discrete state machine. In his *Mind* paper he wrote:

> "The nervous system is certainly not a discrete-state machine. A small error in the information about the size of a nervous impulse impinging on a neuron, may make a large difference to

> the size of the outgoing impulse. It may be argued that, this being so, one cannot expect to be able to mimic the behavior of the nervous system with a discrete-state machine" [424, p. 451].

Turing was aware that the physical processes in the brain are analogue, but believed that the features of the brain relevant to thinking or intelligence can be mimicked by a discrete-state machine. In the *Mind* paper, he then briefly explains how a digital machine can mimic the behavior of an analogue machine.

A Turing machine is a closed system that does not accept input while operating, whereas the brain continually receives input from the environment. Wegner wrote:

> "The claim that interactive systems have richer behavior than algorithms is surprisingly easy to prove. Turing machines cannot model interaction machines (which extend Turing machines with interactive input/output) because interaction is not expressible by a finite initial input string" [453].

Copeland and Sylvan have proposed the *coupled Turing machine* which is connected to the environment via one or more input channels [87]. However, any coupled machine with a finite input stream can be simulated by a universal Turing machine since the data can be written on the machine's tape before it begins operation (note that a machine with a finite lifespan handles a finite amount of input data). Consider the following *Gedanken-experiment*: we've managed to record all of a human's inputs and outputs, received and emitted over a lifetime of interaction; would a universal Turing machine or a Turing neural network now be able to map the inputs to the outputs? In other words, can a human's lifelong behavior be described by a Turing-machine computable function? Copeland wrote:

> "[...] it would—or should—be one of the great astonishments of science if the activity of Mother Nature were never to stray beyond the bounds of Turing-machine computability" [87].

Finally, I would like to clearly emphasize that I am *not* a "believer" of (physical) hypercomputation. Abstract theoretical hypercomputational models are basically irrelevant to computer science and certainly also to machine intelligence since they cannot be built. There is more than good reason to believe that Nature "computes" well below the Turing limit and that the key to machine intelligence does not have to be searched in hypercomputational models, although all this might look promising at a first sight.

2.3 Evolutionary Algorithms

Evolutionary algorithms (EA) [120] are a collection of methodologies inspired by the principles of the biological evolution. The basic concepts go back to the work of Charles Darwin [90]. It is interesting that Turing himself already mentioned "genetical" or "evolutionary" search in his 1948 paper [426, p. 23] in the context of problem solving.

Later, John Holland first introduced and substantiated the idea of *genetic algorithms* (GAs) [197]. To date, many variations and extensions of algorithms and methods inspired by the biological evolution have been proposed. One of the most recent milestones was the introduction by John Koza [224] of a method called *genetic programming* that deals with the automatic generation of computer code. For further reading of general interest, the reader is referred to Banzhaf [28, 120], Michalewicz [266], Bäck [21], Mitchell and Forrest [276], Koza [225], Vose [446], and Fogel [136].

The term evolutionary algorithm usually encompasses a number of related methodologies such as *genetic algorithms*, *evolutionary strategies*, *evolutionary programming*, *genetic programming*, etc.

Mathematically speaking, evolutionary algorithms are a broad collection of optimization methods that are particularly suitable for "hard" problems where little is known about the underlying search space. In order to optimize a solution, an evolution process is simulated, in the course of which the parameters that produce a locally or globally optimal solution are determined. As biological evolution does, evolutionary algorithms maintain a population of *individuals*, each one representing a possible solution for a given optimization problem. Each individual is represented by a finite string of symbols, called *genome*. The *search space*—generally too huge to be exhaustively searched—contains all possible solutions to the problem.

The functioning of a "standard" genetic algorithm can be summarized as shown in Algorithm 1 (pseudo-code):

Algorithm 1 Genetic Algorithm

1: Randomly initialize a population of individuals ($g = 0$)
2: Evaluate the population (fitness evaluation)
3: **while** best or "good" solution not found **do**
4: Selection $g(t) \leftarrow g(t-1)$
5: Crossover
6: Mutation
7: Evaluate the population
8: **end while**

The algorithm starts with the creation of an initial population of individuals. Usually, this population is randomly generated, however, some

heuristics are sometimes applied in order to reduce the search space at the beginning. All individuals are then evaluated according to a certain *fitness function*. The next step is *selection*—according to fitness, and one of many known selection strategies—of the individuals of generation $g(t-1)$ to form a new population. In order to explore the search space, new individuals, i.e., new solutions, are created by means of the *crossover* and *mutation* operators. Crossover simply exchanges—according to some strategies—parts of the parent's genomes to create a child, also called *offspring*. The goal of this operator is to create individuals that move towards the optimal solution. Mutation, on the other hand, randomly changes some symbols on the genome with a small probability. The goal of this operator is to randomly explore the search space.

Compared to local optimization methods, i.e., gradient descent, genetic algorithms have the advantage that they less often get trapped in local minima of the function to be optimized. Since a population of solutions is used, the algorithm can "move away" from local optima if the population finds better solutions in other areas. The disadvantage of genetic algorithms is that there is no guarantee that the process converges to the optimal solution (in reality), although theory suggest that the optimum will be reached with probability 1 for $t \rightarrow \infty$ under some conditions.

2.4 Connectionism and Artificial Neural Networks

From the viewpoint of neural network research, *connectionism* (see for example [32] for a comprehensive overview) is a movement in cognitive science which hopes to explain human intellectual abilities using artificial neural networks.

> There is considerable diversity among connectionist models, but all models are built up of the same basic components: simple processing elements and weighted connections between those elements. [123]

Connectionism has a very long past and one can trace the origin back to the ideas of the early Greek philosopher Aristotle and his ideas on mental associations. However, the birth date and opening shot in neural network research was certainly the 1943 paper by McCulloch and Pitts [261]. They proved that any logical expression could be implemented by an appropriate net of simplified neurons. For this purpose, they assumed that each neuron was binary and had a finite threshold, that each synapse was either excitory or inhibitory and caused a finite delay of one cycle, and that the networks could be constructed with multiple synapses between any pair of nodes.

The *cybernetics* movement and the interest in *self-organizing systems* certainly had an influence on the connectionist movement and vice versa.

Important contributions came from Wiener [460] and Ashby [14,15], but also from McCulloch. Wiener defines cybernetics to be "the science of control and communication in the animal and the machine." The word cybernetics is derived from the Greek *cybernetes*, which means steersman. The cyberneticists wondered if they could make a *thinking machine*, a machine that would be an electrical imitation of the human nervous system. One of their investigations was the creation of the concept of feedback. Astonishingly, Wiener himself, as opposed to Turing, did not believe in machines that can learn.

In the early 1950s, computers, psychology and philosophy tried to join together for the first time. On the one hand, *cognitivism* tried to model the human mind, whereas *connectionism* tried to model the human brain. It was Wesley Clark and Belmont Farley, and not Hebb, who in 1954 first simulated an artificial neural network [75,126]. In 1956, Rosenblatt unveiled his neuron — the *perceptron* — that was principally based on Hebb's ideas [187]. Hebb suggested that a mass of neurons could learn if their connection strengths change according to some rule — today known as the *Hebbian rule*. His idea was that two neurons that are simultaneously active should develop a degree of interaction higher than neurons whose activities are uncorrelated [343].

Towards the end of the 1960s, the proofs on the limitations of simple perceptrons by Minsky and Papert [272] nearly caused the complete abandonment of connectionism. Later Minsky and Papert [273] further developed the perceptron based on the foundations laid by Rosenblatt [346,347].

Artificial neural networks (ANN) have been inspired by the recognition that the human brain processes information in an entirely different way from the classical von Neumann digital computer. The human brain is a highly complex, parallel, and nonlinear information processing machine made up of about 10^{11} neurons whereas each neuron is connected to 10^3 to 10^4 other neurons. The information is stored in the contact points between different neurons, the *synapses*.

An artificial neural network is an information processing system which is made up of a number of simple, highly interconnected processing elements—the *neurons*—which process information in parallel. As a simplification, the neuron might be considered as a sort of *detector* that detects the existence of some set of conditions and that responds with a signal that communicates the extent to which those conditions have been met [308]. Artificial neural networks can be considered as an alternative approach to the problem of computation just as biological neural networks are one of many possible solutions to the problem of processing information. Most often, artificial neural networks are "neural" only in the sense that they have been inspired by *neuroscience* but not necessarily because they are faithful models of biologic neural and cognitive phenomena. For example, connectionists usually do not attempt to explicitly model the variety of different kinds of brain

neurons, nor the effects of neurotransmitters and hormones. Even today, connectionist networks are designed most often by hand and reflect important theoretical claims, experience, and knowledge on the part of the modeler. There are very little general rules on how to design connectionist models. The network's architecture and topology is often the most important part. Elman stated: "[...] part of what a network knows lies in its architecture" [123]. Nevertheless, artificial neural networks are a broadly accepted and viable computational model for a wide variety of problems.

Figure 2.6 shows the structure of an abstract neuron [343]. Each input has an associated weight. The input value x_i is usually multiplied by the weight w_i. The neuron's output is computed with a *primitive function* that can be selected arbitrarily. Thus, "[...] artificial neural networks are nothing but *networks of primitive functions*" [343]. Many different neuron models, primitive functions, network topologies, timing characteristics, etc. have been proposed since the model of McCulloch and Pitts [261] and Turing's unorganized machines.

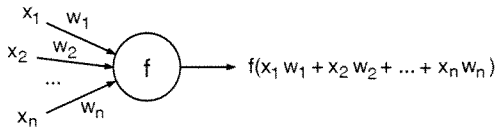

Figure 2.6: An abstract neuron. Each input has an associated weight. The input value x_i is usually multiplied by the weight w_i. The neuron's output is computed with a function that can be selected arbitrarily.

Learning is the ability of the brain to adapt its behavior to a changing environment. Normally, this ability is used to improve the performance of the system, e.g., of a human being or animal in its environment. As Simon Haykin stated, "the process of learning is a matter of viewpoint, which makes it all the more difficult to agree on a precise definition of the term" [185]. For example, learning viewed by a psychologist is quite different from learning in a classroom sense. In this book, only the following definition of learning is considered:

The words "learning" and "adaptation" tend to mean different things to different people. According to the Webster's New Collegiate Dictionary (1975), "adaptation" is defined as following:

Definition 2.4.1 (Adaptation)
1: the act or process of adapting: the state of being adapted 2: adjustment to environmental conditions: as a: adjustment of a sense organ to the intensity or quality of stimulation b: modification of an organism or its part that makes it more fit for the existence under the conditions of its environment. ■

In contrast, "learning" is often seen as "knowledge or skill acquired by instruction or study". Therefore, to learn can be defined as "to gain knowledge or understanding of or skill by study, instruction or experience" (Webster's New Collegiate Dictionary, 1975)

From the perspective of computer science, learning might also be considered as what an entire system does while adaptation mainly refers to the progressive changes a system undergoes. In the context of neural networks, learning is however often considered as stated in the following definition. In summary, the reader should only be aware that there are different ways the words "learning" and "adaptation" tend to be used, but I don't try to further define or redefine them.

Definition 2.4.2 (Learning in the context of neural networks)
Learning is a process by which the free parameters of a neural network are adapted through a process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes take place [185, p. 50]. ■

Learning in artificial neural networks can be regarded as a search for parameters (weights, thresholds, switches, etc.) that optimize a predefined function (input-output mapping). Learning in artificial neural networks is hard and requires exponential time irrespective of the learning algorithm used. Judd [208, 209] showed that the learning problem in neural networks is NP-complete, even for approximative learning. Thus, in the worst case, one will not be able to do much better than just randomly exhausting all combinations of settings to see if one happens to work. However, these theoretical results do not rule out the possibility of finding a polynomial-time algorithm for the training of certain classes of problems [184].

Learning algorithms can be classified into two main classes (see also Figure 2.7) [343]:

- *supervised learning*
- *unsupervised learning*

The basic difference between these two learning modes concerns whether the net uses an external report (from the supervisor) to modify its performance. Supervised learning basically relies on three things [69]:

1. input,
2. the net's internal dynamics, and

3. an evaluation of its weight-setting job.

On the other hand, unsupervised learning relies on two things only:

1. input, and
2. the dynamics of the net.

No external report on its behavior vis-a-vis its weight-setting progress is provided.

> "In either case, the point of the learning algorithm is to produce a weight configuration that can be said to represent something in the world, in the sense that when activated by an input vector, the correct answer is produced" [69].

Unsupervised learning is normally used when, for a given input, the exact output the network should produce is unknown. Supervised learning is further divided into methods which use *reinforcement learning* or *error correction*. Reinforcement learning is used when after each presentation of an input-output example we only know whether the network produces the desired result or not. The weights are updated based on this information (that is, the boolean values *true* or *false*) so that only the input vector can be used for weight correction. In learning with error correction, the magnitude of an error, together with the input vector, determines the magnitude of the correction to the weights.

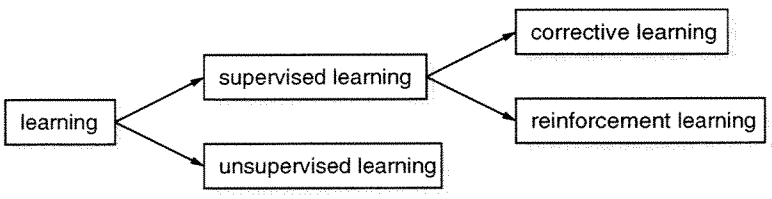

Figure 2.7: Classes of learning algorithms.

One of the simplest learning rules that can be used is *Hebbian learning*, proposed in 1949 by the psychologist Donald Hebb [187]. His idea was that two neurons which are simultaneously active should develop a degree of interaction higher than those neurons whose activities are uncorrelated [343]. During learning both input and output are clamped to the network and the weight update is given by:

$$\Delta w_{ij} = \eta x_i x_j \qquad \text{The Hebb rule} \tag{2.2}$$

The factor η is the learning constant. Figure 2.8 illustrates the Hebb rule.

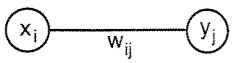

Figure 2.8: Illustration of the Hebb rule.

This rule is also referred to as *Hebbian plasticity*. Hebb gave this formulation in 1949:

> "When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased" [187].

As a consequence, the increase in strength in some synapses must be compensated for by a decrease in others. Only the more successful synapses can grow; the less successful ones weaken and eventually disappear [442].

One synapse on its own cannot efficiently produce favorable events. For that it needs the cooperation of other synapses. The rules of cooperation and competition act on a local scale and are able to produce ordered connection patterns. However, they do not necessarily organize the nervous system for optimal biological utility [442]. Malsburg also concluded that central control as the only criterion for growth or decay of synapses is not sufficient. This is simply a problem of scale. A central controller would not have enough time to regulate the strength of all the synapses of the human cerebral cortex.

Reinforcement learning is learning what do to, i.e., learning how to map situations to actions, so as to maximize a numerical reward signal. The learner is not told which actions to take, as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them. Reinforcement learning is thus defined not by characterizing learning methods, but by characterizing a learning problem [397]. Reinforcement learning is different from supervised learning, the kind of learning studied in most current research in machine learning and artificial neural networks. Supervised learning is learning from examples provided by a knowledgeable external supervisor. However, it is not always possible to obtain examples of a desired behavior that are both correct and representative for all situations in which the system has to act. For further reading about reinforcement learning, I highly recommend Sutton and Barto [397].

Finally, I'd like to mention that connectionist models are examples of *subsymbolic* (or *subcognitive*) architectures [73] (in contrast to *symbolic* architectures). Smolensky [383] characterizes the difference between the symbolic and subsymbolic paradigms as follows (adopted from [195]). In the symbolic paradigm, descriptions used in representations of situations are built of entities that are *symbols* both in semantic and the syntactic sense. In the subsymbolic paradigm, such descriptions are built of *subsymbols*, i.e.

fine-grained entities (such as nodes and weights in connectionist networks or classifiers in a classifier system) that give rise to these symbols.

A wide variety of neural network literature is available and every year an enormous amount of papers is published in international journals and on conferences. Rojas' book [343] is a good introduction for beginners whereas Haykin [185] offers an excellent and comprehensive foundation for the mathematically interested expert. Arbib's "Handbook of Brain Theory and Neural Networks" [13] offers an excellent encyclopedic collection of papers that cover almost every domain in neural network research and brain-level theories. For readers interested in cognitive neuroscience and biologically-inspired neural networks, a good starting point might be O'Reilly and Munakata's book [308] or Churchland's "The Computational Brain" [69].

2.5 Evolutionary Artificial Neural Networks

Today, a lot of work is focused on the understanding of the interaction of learning and evolution in biological systems. From an engineering point of view, the combination of learning and evolution shows that significant advantages can be gained in the adaptation of systems to an environment for a given task [4, 300].

Evolution and learning are two forms of biological adaptation that differ in space and time. Evolution is a selective reproduction and substitution of a population of individuals. Learning, instead, is a set of modifications taking place within each single individual during its own lifetime. Often learning also refers to the modification of synaptic weights of a neuron network during its *lifetime*. Both forms of adaptation have been successfully combined in many applications and it has been shown that, although they are two distinct forms of adaptation, they strongly influence each other [296].

Evolutionary artificial neural networks (EANN) refer to a special class of *artificial neural networks* (ANNs) in which evolution is another fundamental form of adaption in addition to learning. Evolutionary algorithms are used to perform various tasks, such as connection weight training, architecture design, learning rule adaption, input feature selection, connection weight initialization, etc. EANN can adapt as well to an environment as to changes in the environment [468]. In a broader sense, EANNs can be regarded as as a general framework for adaptive systems, i.e., systems that change their architectures and learning rules appropriately without human intervention.

Yao classified evolution in ANNs in three different levels [468]:

1. connection weights,
2. architectures, and
3. learning rules.

The evolution of connection weights is an adaptive and global approach to training, especially in the reinforcement learning and recurrent network learning paradigm where gradient-based training algorithms often experience great difficulties. The evolution of architectures enables ANNs to adapt their topologies to different tasks without human intervention and thus provides an approach to automatic ANN design, as both ANN connection weights and structures can be evolved. Finally, the evolution of learning rules can be regarded as a process of "learning to learn" in ANNs where the adaption of learning rules is achieved through evolution.

More recent approaches try to encode the type of learning used for each neuron instead of the synaptic weights on the genome. This is more biologically plausible and the final system is more robust to environmental changes.

When applying GAs to neural network design, each *genome* (also called *string* or *chromosome*) generally encodes a network architecture, the network weights, or both. The encoded network architecture is known as *genotype*. In analogy with natural genetics, the actual network architecture realized by a genotype is called *phenotype*, the structure that emerges as the result of interpretation of the genotype. The central topic of this section is the *genotype-to-phenotype mapping* on which Langton expressed:

> "[...] the phenotype is a nonlinear function of the genotype, and the label of that nonlinear function is *development*" [234].

Miller *et al.* [267] have classified encoding techniques into two categories:

1. *strong specification schemes*, also called *direct encoding*, or *blueprint encoding*, and
2. *weak specification schemes*, also called *indirect encoding*.

They define the mapping from genotype to phenotype in terms of the level at which the encoding takes place. A weak network specification is defined as one which only considers structural elements at the highest level, namely the organization of layers, nodes, but not individual connections. The indirect encoding scheme is used in order to reduce the length of the genotypical representation. A strong specification scheme is a low level form of encoding in which every connection is specified individually within the genotype.

Roberts and Turega [341] proposed a third, the *intermediate representation scheme*, also known as *grammar encoding scheme* [218]. This scheme makes use of fractal techniques like L-systems developed by Lindenmayer [239] and extended by Prusinkiewicz [327]. The process may be compared to the growth of natural objects. For more details about connectionist modeling using L-systems, see for example Vaario [432].

In general, the way in which the phenotype-to-genotype coding should be realized is not straightforward. In most current models, the representations of the genotypic and phenotypic forms coincide, that is, the inherited genotype directly and literally describes the phenotypical neural network. This approach encounters problems of scalability [218] because the number of bits of information necessary to encode a network increases in general exponentially with the number of neurons of the network. Note that this is not the case with Turing neural networks as the number of connections to encode is linearly dependent on the number of network neurons.

Roberts and Turega [341] conclude that the encoding method of a network indeed influences the quality of the networks produced. On small networks, containing only a few nodes, the strong encoding scheme was found to outperform the other encoding schemes. The grammar encoding scheme appears to be relatively unaffected by the problem size.

Yao concludes in [468] that the direct encoding scheme of ANN architectures is very good at fine tuning and generating a compact architecture. The indirect encoding scheme is suitable for finding a particular type of ANN architecture quickly. As with every evolutionary algorithm, the search for an optimal solution is usually very computationally expensive. It is thus better not to employ EAs at all possible levels (architecture, weights, learning rules) of evolution.

For further reading about encoding strategies, see also Hancock [178] and Gruau [171], for an overview on the evolutionary design of artificial neural networks see for example de Garis [97], Yao and Liu [469, 470], Yao [468], Nolfi and Floreano [296, 297], and Nolfi and Parisi [299], and many others.

2.6 Some Developmental Models

Biological development is a highly complex, hierarchical program that operates across many different scales and at each level of scale many different mechanisms and self-organizing processes are being involved. The dynamic interactions between the different mechanisms and elements result in a highly complex system. Development in artificial systems is a very important issue and it is commonly and for a long time already believed that it is a key to the generation of highly complex systems (see for example Kitano [219]).

Much of the early work in developmental modeling focused on modeling a particular mechanism only. The mechanism has then be considered in isolation and not in interaction with other mechanism, although it has long been evident that the interactions of different mechanisms are critical to several issues, like for example pattern formation. Today, there is a growing trend towards developmental models based on multiple mechanisms (like cell migration, cell division, attraction forces, environmental influences, etc.). In the following, some of the most important models and work shall be

presented.

Besides D'Arcy Thompson's classical work "On Growth and Form" [414], first published in 1917, Alan Turing was one of the first persons interested in modeling morphogenesis. In his 1952 seminal paper entitled "The Chemical Basis of Morphogenesis" [355, 425] he proposed a mathematical theory of cell-cell interaction via chemical substances (also called *reaction-diffusion model*). Turing described several sets of differential equations that governed the interactions between different substances that diffuse and react with each other. The original intent of the reaction-diffusion model was to explain the "breakdown of symmetry and homogeneity" or the emergence of a pattern in an originally homogeneous medium.

In 1968, Aristid Lindenmayer developed a grammar-based technique called L-systems [239] (further developed later by Prusinkiewicz [327]). L-systems is a rule rewriting formalism that allows to efficiently describe growth, e.g., plant growth. For a particular L-system, the growth always starts from the same seed cell, called *axiom*. *Production rules* are used to describe the growth of new cells from the old cells. L-systems are often used to model development in combination with evolutionary algorithms and neural networks, are computationally feasible models, but not well suited for models based on local interactions only. Kitano, for example, uses a graph-generation grammar based on L-systems where the structure of the network is not directly encoded on the genome [218]. His approach shows better scaling properties, is more biologically plausible, and generates more complex networks, but it is not always easy to determine whether a grammar-based encoding can express every possible network architecture. L-systems have also been applied to neural networks [218, 432] and to modeling cell layers [95, 96, 327].

A further work that uses L-systems [239] is described in Boers and Kuiper's master's thesis [46]. It combines a genetic algorithm and a learning process with a L-system grammar that models development. The genetic algorithm operates on a set of production rules that are applied to an axiom for a number of iterations. The resulting string is then transformed into a structural specification for a classical feed-forward neural network. In the next step, the network's weights are trained by a simple back-propagation algorithm. The genetic algorithm uses the fitness estimate provided by the back-propagation algorithm. The model has for example been applied to classify handwritten digits on a 5×5 grid.

Another important grammar-based approach comes from Gruau [170, 171, 174]. He developed a special encoding scheme called *cellular encoding*. Instead of rewriting characters, the rewriting grammar directly rewrites neurons. A genetic algorithm can then be used to find a grammar tree suitable for a given problem. In order to speed up the search for functional networks, Gruau also included several forms of learning in his model (see for example [174]). Within the framework of cellular encoding, a cell is a node

of an oriented network graph with ordered connections. Each cell carries a duplicate copy of the chromosome, i.e., the grammar tree. The cell has an internal reading head that reads at a different position from the grammar tree. The character symbols in the grammar tree represent instructions for the cell development. In some way, the cell is thus similar to a Turing machine [422]. Gruau also refers to the grammar tree as a *program* and to the characters as a *program symbol*. A cell also contains some internal registers. Some are used during development, others determine the weights and the thresholds of the final neural network. The developmental process always starts from a single cell—the *ancestor cell*—connected to an input and an output pointer cell.

Gruau evolved for example a neural network capable of controlling the locomotion of a six-legged animat. The same problem has already been solved by Beer and Gallagher [35], however, they took advantage of the various symmetries that such a controller is supposed to exhibit. Gruau did not help the evolutionary algorithm, but solved a slightly simpler version of the problem.

Note that Gruau's cellular encoding approach is currently having a revival in the BLOB computing project as presented on page 78.

Grammar based systems have also been enhanced to include physical behavior (i.e., forces) and the modeling of cell layers. Odell *et al.* [304] studied in 1981 the mechanical aspects of morphogenesis. They presented a model for the folding and movement of cell sheets and applied it to certain elementary types of gastrulation and neurulation. Work with mechanical models has been continued later with more detailed and more complex models. Cells have different shapes and various forces are applied. Oster *et al.* [304] and Weliky and Oster [457], for example, use cells that have a polygonal shape in three dimensions. Cells are subjected to forces that are due to osmotic pressure and to the elastic membrane.

Murray's mechanochemical model [279, Chapter 17] is based on the assumptions that cells can migrate and that cells can generate large traction forces. The mechanical part of the model is then combined with a reaction-diffusion model.

The mechanical morphogenesis (morphomechanics) idea is also at the basis of the work of Lev Beloussov, Dick Gordon, and others. Their ideas (the embryonic development is driven and controlled by mechanical forces) are, however, in most parts completely opposed to the theories (timing and positional information of cells) of Lewis Wolpert.

2.7 The POE Model for Classifying Bio-Inspired Machines

The POE model has been proposed as a mean to classify biologically-inspired computing machines and especially hardware. In the following the basic ideas shall be described (the text is a slight modification of [377]; see [376] for a detailed presentation of the model).

Biological systems grow, live, adapt, and reproduce, characteristics that are not truly encompassed by any existing computing system. The concept of "living" has a number of consequences in terms of adaptation, interaction with the environment, and the ability to deal with limited resources. Methodologies and technologies that enable the construction of artificial systems that "live", grow, adapt, and reproduce in hardware would allow a quantum leap in performance for many computing systems known so far.

If one considers life on Earth since its very beginning, three levels of organization can be distinguished [354, 376, 377]:

- **Phylogeny:** The first level concerns the temporal evolution of the genetic program, the hallmark of which is the evolution of species, or phylogeny. The multiplication of living organisms is based upon the reproduction of the program, subject to an extremely low error rate at the individual level, so as to ensure that the identity of the offspring remains practically unchanged. Mutation (asexual reproduction) or mutation along with recombination (sexual reproduction) can give rise to the emergence of new kinds of organisms. The phylogenetic mechanisms are fundamentally nondeterministic, with the mutation and recombination rate providing a major source of diversity. This diversity is indispensable for the survival of living species, for their continuous adaptation to a changing environment, and for the appearance of new and better adapted species.

- **Ontogeny:** Upon the appearance of multicellular organisms, a second level of biological organization manifests itself. The successive divisions of the mother cell, the zygote, with each newly formed cell possessing a copy of the original genome, is followed by a specialization of the daughter cells in accordance with their surroundings, i.e., their position within the ensemble. This latter phase is known as cellular differentiation. Ontogeny is thus the developmental process of a multicellular organism. This process is essentially deterministic: an error in a single base within the genome can provoke an ontogenetic sequence which results in notable, possibly lethal, malformations.

- **Epigenesis:** An given ontogenetic program (e.g., the human genome) is limited in the amount of information that can be stored. In most

organisms, the information that can be stored on the genome is not sufficient to specify the complete organism (although this is not a fundamental problem but rather a choice Nature has made). A well-known example is that of the human brain with some 10^{10} neurons and 10^{14} connections, far too large a number to be completely specified in the four-character genome of length approximately 3×10^9 . Therefore, upon reaching a certain level of complexity, there must emerge a different process that permits the individual to integrate the vast quantity of interactions with the outside world. This process is known as epigenesis and primarily includes the nervous system, the immune system, and the endocrine system. These systems are characterized by a basic structure that is entirely defined by the genome (the *innate* part), which is then subjected to modification through the lifelong interactions of the individual with the environment (the *acquired* part). The epigenetic processes can be loosely grouped under the heading of *learning* systems.

In analogy to nature, the space of biologically-inspired systems can be partitioned along the same three axes: *phylogeny*, *ontogeny*, and *epigenesis*. We refer to this as the *POE* model (Figure 2.9).

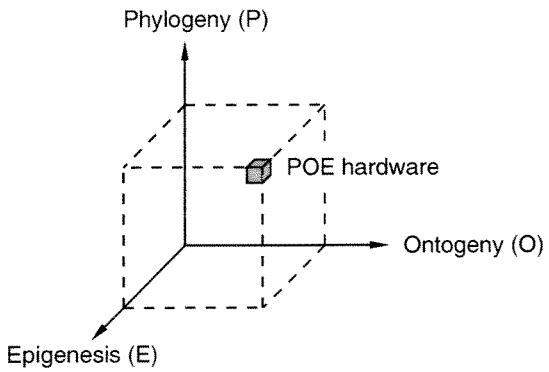

Figure 2.9: The POE model. The combination of the three POE axes gives birth to novel bio-inspired computing systems, i.e., POEtic machines that combine evolution, growth (cellular development), and learning.

The POE model — originally intended to classify biologically-inspired hardware — only considers three forms of adaptation and organization and therefore presents a limited view of Nature and biological(ly-oriented) systems.

There are a certain number of questions which arise and which I'd like to briefly address here:

- Does the POE model offer a valid and useful classification of biologiclaly-inspired machines?
- What do the axes mean in the model? What does mean "a machine is more phylogenetic than another"? What are the measuring units?
- Is there anything the model or classification can predict? If yes, what?
- Is the model open for new developments and findings? How does it deal with new machines. For example, where would one classify reaction-diffusion systems?
- Is the construction of POE machines really the ultimate goal or is this an purely academic goal? What are the benefits of such machines? What are the benefits of a hardware implementation?

Without providing an answer to all questions listed above, I would like to emphasize several points. First of all, classifications are already a much debated and controversial field in biology. For example, where would one put a network of mushrooms as it depends on the point of view where the individuals and the population lies. Is a single mushroom to be considered as an individual or should the entire network be considered as an individual as the mushrooms could not survive without it? And what about certain trees which possess cells with different genomes?

The POE model to classify biologically-inspired machines takes this already problematic classification and applies it to computer science. The risk with many classifications is that they might obscure the view on the real foundations of systems. Seeing the world through POEtic glasses allows to classify virtually any machine in the POE model, however, does this really make sense?

The three classes (i.e., P, O, E) or axes are fairly large and unprecisely defined. The result is that almost any system might be classified according to the POE model in one or another way — and that's exactly one of the problems of the model: its scientific usefulness is limited and the model does not allow for predications. The principal point, however, which somehow limits the model's usefulness, is the following: although the graphical representation contains axes, no scale is available. Where should a certain phylogenetic machine be placed in the three dimensional space, for example? What means "a machine is more epigentic than another"? In order to feel the difficulty, the reader might try to position "genetic programming" [224] and "evolution strategies" [43]. Both lie on the phylogenetic axis, but where? Or where is a machine to be classified that dynamically modifies (e.g., rearranges) the genome during the developmental phase? Sipper *et al.* [376] provide some rather vague cues by sub-dividing the axes into several classes,

but this does not seem to be a fully convincing solution to me and this issue therefore remains somehow obscure to me.

In his recent book, Sekanina gives a cue: "Systems that show a higher degree of hardware implementation and a deeper level of inspiration in biology are plotted upward along a given axis" [363, p. 5]. This is even more metaphysical and both criterion can, but are usually not at all related! I do not claim to have found a well thought out solution, but I simply wondered whether it would not be more appropriate to use classes instead of axes actually. An illustration is reproduced in Figure 2.10. It would be possible to define very clear criterion for each class end to provide a fairly unambiguous classification then.

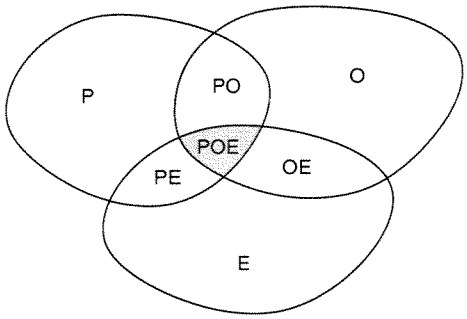

Figure 2.10: POE classes instead of POE axes?

Another very important aspect is time, which is in my opinion too much neglected in the POE model. From a complex systems point of view, for example, one might say that the three axes all possess the same underlying mechanisms, but on different time scales. Figure 2.11 shows the typical time scales the three forms of adaptation are working in. For many applications, a simple classification by means of such a diagram would be sufficient. However, the classification in terms of adaptation time might also very well complete a classification based on three classes or axes.

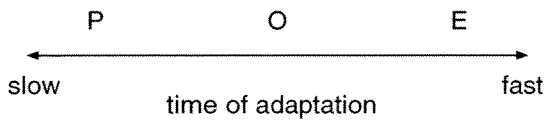

Figure 2.11: The three forms of adaption usually operate in different time scales, but this is not necessary the case.

In summary, the model contains nothing new but only brings in new terms for well established expressions. It is therefore not astonishing that most people working in the domain of biologically-inspired machines simply

make use of the traditionally used terms, such as *artificial neural networks*, *evolutionary algorithms*, *development*, etc. Finally, Sipper *et al.* provide the answer to most of my considerations in 1997 already:

> In fact, the success of the "POE community's" endeavor can be measured by the disappearance (extinction?) of the POE model (or rather its utility). This will ensure from the composition of future systems which may eventually exhibit characteristics of all three axes, i.e., reside within the POE space. At such time, the POE model will have outlived its usefulness. [376, p. 93]

As we will see in the next section, many already existing machines come close to a POEtic machine.

2.8 Towards POEtic Machines

Development, evolution, and learning are forms of adaptation that allow individuals and species to adapt to a changing environment on different time scales. Sometimes, this adaptability is even considered as "intelligence" *per se*. In recent years, there has been a great interest in combining learning and evolution with artificial neural networks (see Section 2.5).

The main testimony of this section is that systems combining learning, evolution, and developmental processes were made well before the "introduction" of the POE model presented in the previous section, although its is regularly stated in the literature that the ultimate goal should be to build machines (in hardware)—so-called *POEtic* machines—that combine the three axes.

In nature, the development of a nervous system can roughly be separated into three phases: (1) first, nerve cells are generated through cell division; (2) the cells grow out axons and dendrites along specific routes in order to form a provisional interconnectivity; (3) last, the synaptic interconnections are refined and remodeled according to the pattern of electrical activity in the neural network. The final phase normally continues into an individual's adult life. Most actual neural network and evolutionary neural network approaches don't model the three above mentioned phases together but are restricted to the one or two phases only.

Evolutionary systems and hardware are often classified in two categories: (1) *intrinsic* and *extrinsic* hardware. Extrinsic hardware simulates evolution by software (*offline evolution*) and only downloads the best configuration to hardware in each generation. On the other hand, intrinsic hardware simulates evolution (*online evolution*) directly in hardware, i.e., each genome is used to (re-)configure the hardware. Since a single individual in nature

does not evolve itself, extrinsic evolution seems more biologically plausible. Therefore, most often, the phylogenetic axis is sufficiently well represented by encoding the entire system on a genome. It is then rather easy to apply offline evolution (extrinsic hardware).

The following sections shall give a short — and certainly incomplete — overview on some of the most promising and most important models trying to put together evolutionary principles, growth processes, and artificial neural networks. For a short introduction to development, learning, and evolution in animates, the interested reader is also referred to the paper of Kodjabachian and Meyer [221].

2.8.1 A Hexagon Based Tissue

The diploma thesis of Jens Astor [18] and further work in collaboration with Chris Adami [19] presented a model of decentralized growth and development of artificial neural networks inspired by developmental biology and the physiology of nervous systems. The model is particularly interesting since each neuron is completely autonomous and its behavior is determined only by the genetic information contained within the cell and by the local concentration of substrates. An artificial chemistry is used to model the chemicals and substrates. The model implemented follows the four basic principals of molecular and evolutionary biology [19, p. 190]:

- *Coding.* The model should encode networks in such a way that evolutionary principles can be applied.
- *Development.* The model should be capable of growing a network by a completely decentralized growth process, based exclusively on the cell and its interactions.
- *Locality.* Each neuron must act autonomously and be determined only by its genetic code and the state of its local environment.
- *Heterogeneity.* The model must have the capability to describe different, heterogeneous neurons in the same network.

Astor and Adami state that "[o]ne of the key features of a model implementing the above principles will be the absence of explicit activation functions, learning rules, or connection structures. Rather, such characteristics should emerge in the adaptive process."

The model they used is two-dimensional surface ("tissue") made up of hexagon unit (see Figure 2.12). Hexagon units have bee chosen in order to alway have the same distance to each neighboring unit. Each hexagon harbors certain chemical concentrations of substrates. The tissue is surrounded by a special boundary that absorbs the substrates and thus models the diffusion in infinite space. Four different classes of substrates are distinguished:

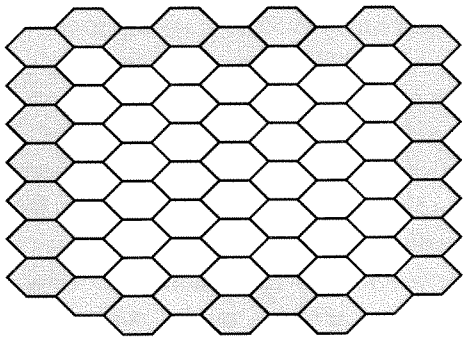

Figure 2.12: Tissue made up of hexagon units with boundary elements that absorb substrates.

external, *internal*, *cell-type proteins*, and *neurotransmitters* (for more details see [18, 19]). Furthermore, the tissue can harbor different neural cell types (*actuator cell*, *sensor cell*, and common neurons) and connections. Common neurons represent hidden neurons whereas actuator and sensor cells represent the input and output layer of the network. There are no special connection cells. A hexagon unit that contains a neurons also contains dendritic and axonic connections to other cells. The growth of axons and dendrites is guided to their target by means of diffusible growth factors. Each neural cell contains a genome that encodes its behavior. An important aspect of the model is that genes are regulated and regulate each other. A neuron's function is mainly determined by its genetic information and the local concentrations of substrates. In order to allow for learning, the model also assigns for each dendritic connection a weight for each type of neurotransmitter. The weight can be modified by different expression commands. A simulation starts be creating a number of sensor and actuator cells that depend on the complexity of the problem. Then, a single initial neuron is placed in the middle of the grid. The neuron then starts growing axons and dendrites, produces offspring cells, and might initiate cell differentiation.

Astor and Adami's model has successfully been applied to some very simple problems and they agree on the fact that "[...] the system has to be evaluated in more depth" [19] and that experiments on a large scale have to be done. As with many other artificial life model, the number of free parameters is high, the genotypes are large, and the search space thus huge. With the current computational power, only very simple problems can be considered[2]. It has been shown that the model is able to give rise to the development of artificial neural networks based on local interactions only. The existing model could be improved in considering cell death as well as morphology. Cells are currently always located at the same place from their

[2]The authors propose, however, an asynchronous and distributed system that allows to search for good genomes in parallel on the internet: `http://norgev.alife.org`.

birth on and cannot move along a chemical gradient. In order to make to model more biologically plausible, one might also consider including cell membranes in the model.

2.8.2 A Multiple-Developmental Model

The importance of dynamic morphology in neural-like systems has been demonstrated in the Ph.D. thesis of Kurt Fleischer [132] in 1995 (an early version of the work was published in 1994 [133]). Fleischer introduces a model of multicellular development that combines elements of the chemical, cell lineage, and mechanical models of morphogenesis pioneered by Turing [355, 425], Lindenmayer [239], and Odell [304]. Cell migration is an important aspect of development, especially of the neural development. His model—and this distinguishes it from previous work in reaction-diffusion systems and grammar based systems—allows cells to move freely within the environment. The internal state of each cell in the model is represented by a time-varying state vector that is updated by differential equations. The driving question behind Fleischer's work was the following [132]: "What features of a representation scheme and a developmental process are important for describing a wide range of structures, and also provide the robustness and developmental properties?"

He was able to show by means of different simulation experiments that his model represents a wide range of biologically-relevant phenomena in two and three dimensions. Among other applications, he also applied his model to the evolution of artificial neural networks using a developmental model, however, he too failed to show revolutionary results as there are too many parameters in his model. His largest developmental model—made up from 70 cells—was able to follow gradients of diffusion chemicals. Fleischer also applied his model to synthetic biology with the main goal to study multicellular morphogenesis and pattern formation.

Let's take a closer look into Fleischer's model. He identified the following features of biological development as being critical to morphogenesis and pattern formation [132]:

- *Boundary:* A cell has a boundary surface with some shape and location.
- *Chemicals:* Chemicals exist within the cell, on the cell surface, and in the extracellular space.
- *Sensors:* A cell can sense local information from the extracellular environment, and the proportion of its surface chemicals which are bound to complementary chemicals.
- *Cell lineage:* A cell can control the orientation of its next cleavage, as well as the initial states and cell types of its immediate children.

- *Equations of motion and shape change:* A cell is capable of exerting forces to move and change shape, which are mediated by viscous dynamics, collisions with other cells and obstacles, and adhesion due to binding surface chemicals.

- *Death:* A cell can cause its own death.

In order to overcome difficulties in modeling the system, Fleischer has chosen to model cell shapes with circles or spheres instead of an arbitrary shape. Furthermore, each intercellular chemical is represented by a single value and each surface chemical is assumed to be uniformly distributed on the boundary of the cell. From this features and abstractions, the model has been built. Each cell has an array of state variables that represent its intercellular and surface chemicals. State changes are effected using differential equations, the *cell state equations.* Input from the cell's sensors are parameters to the cell state equations. The cell's behavior is determined by the *cell behavior functions.* Cells are only able to access local information and do not know their absolute position and orientation in the world. Their behavior is only affected by their environment. Fleischer has also implemented a simple abstraction of neural growth using growth cones witch are modeled as small cells with few restrictions. The growth cones are connected to the parent cell by a neurite and growth cones and cells communicate via a set of state variables which are held in the neurite. It is also possible to simulate spiking neurons.

Fleischer's approach revealed also several limitations. For example, the abstraction of the spherical cell shape is rather unrealistic and a more detailed cell shape model is desirable. Nevertheless, the thesis presents a multiple-developmental model that can represent a variety of biological phenomena. "The next step is to apply the model to a particular biological system" [132].

In the same context, Mjolsness' connectionist model [278] is similar to Fleischer's approach [132] since it also has a differential equation model of gene expression, a representation for cell differentiation, and mechanisms for cell-cell interaction. However, Mjolsness' model uses a special-purpose geometric model whereas Fleischer uses a more general geometric mechanism using cell collision, cell adhesion, and diffusion chemicals.

The *Cell Programming Language* [6] is a combination of cellular automata, lattice models, and discrete event simulation. Compared to Fleischer's model [132], the cell programming language supports all features, at the exception of the extension to neural structures. The representation, however, is quite different (see [132]).

2.8.3 Agents are going Cellular

Another biologically-defensible model of development that also includes evolutionary techniques and neural network-like structures has been proposed by Frank Dellaert and Randall Beer [102–104]. They were mainly interested in the synthesis of autonomous agents—including bodies and control systems—without using a direct phenotype to genotype mapping since this is more biologically plausible. The first principal component of Dellaert and Beer's approach is that gene expression is regulated by a *genetic regulatory network*. The cell state is represented as a binary vector which is updated via a boolean network à la Kauffman [213]. The second component of their work consists of a simple two-dimensional cellular simulator that models the cellular level. Each cell is represented by a square element that can divide in any of the two directions (vertical or horizontal). The last aspect taken into account in their model concerns the intracellular communication which is extremely important in biological development. Thus, the future state of a cell will not only depend on its own state but also on the state of the surrounding cells.

A simulated cell cycle consists of two phases: (1) interphase and (2) mitosis. During the interphase, the cell state is synchronously updated until a steady state is reached. During mitosis, it either stays intact and waits for the next interphase or the cell divides into two daughter cells whereas its state vector is inherited by the two daughter cells. The daughter cells then differentiate into distinct cell types (i.e., sensor, actuator, or control-neuron cells). No cell movement has been implemented in the model. The organism itself is a two-dimensional square of cells. Development starts from a single cell (square), the zygote, then subsequently divides according to the state of the genetic regulatory network (see Figure 2.13).

Dellaert and Beer have successfully shown that their model is evolvable and that it can be optimized to perform some task in the fully developed organism (e.g., an obstacle avoidance task [104]). However, is has been impossible to evolve complex organism from scratch. It should also be noted that in general, no learning takes place in their model (they did, however, without giving any details, some experimentation in [104]).

2.8.4 An Artificial Retina

In 1998, Alistair G. Rust investigated in his Ph.D. thesis [351] models inspired by the biological neural development. The model he chose as appropriate develops three dimensional neuron-to-neuron interconnections. Rust's ultimate goal is having an artificial neural network model which could automatically bootstrap to a given application in developing networks of arbitrary size, connectivity, and functionality. As a testbed application, the development of an artificial edge-detecting retina has been chosen. An arti-

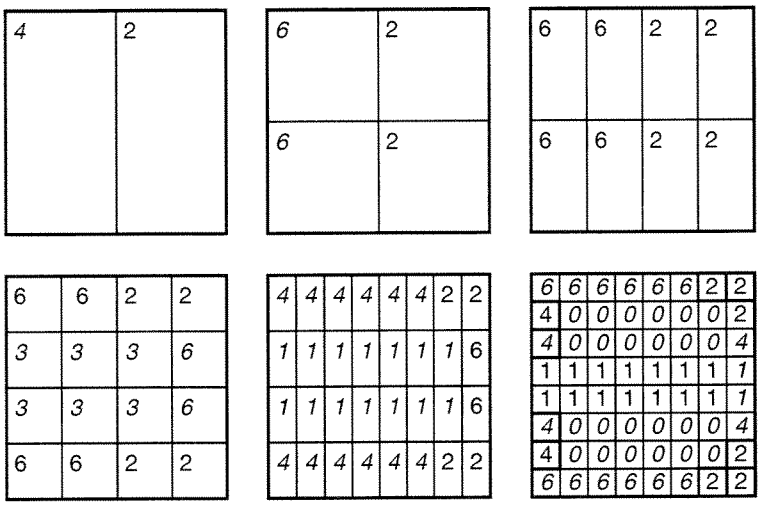

Figure 2.13: The six consecutive stages of development in one of Dellaerts organisms. Digits denote cell types, italics indicate that in the corresponding cell the color/cell type changed relative to the previous developmental stage. Source: [102, p. 91].

ficial retina model presents a number of advantages. The network is considered to be hardwired and no learning and feedback mechanisms are required. The structure is modular and arranged in layers. For edge-detection, only three types of neurons are used: (1) cones, (2) bipolars, (3) and horizontal neurons. And last but not least, the retina has been chosen since it is a widely studied model and thus provides exemplar solutions.

The neurons used are stationary in the environment and are able to grow out axons and dendrites. The growth process is guided by a set of abstracted developmental rules that each neuron contains. Neurons, axons, and dendrites emit local chemical gradients of artificial neurotrophin. Rust states that "[t]he design of the outgrowth rules is motivated by Stryker's assertion that the development of precise structure of the brain can be governed by imprecise rule [395]." Parameters allow the developmental phase of the network to be flexible and adaptive. A genetic algorithm has been used to search through the developmental parameter and to evolve an artificial retina.

In summary, the networks built do not bear strong resemblance to classical artificial neural architectures. The model, however, incorporates interactive self-organization during the developmental phase and thus creates complex three-dimensional neural structures using relatively few and simple rules. In addition, a novel activity-based pruning mechanism has been realized.

Rust has shown that a simple artificial edge-detecting retina can be

evolved using genetic algorithms to identify sets of optimal parameters. The network's connectivity self-organized by means of interactive neurite branching, self-regulated outgrowth rates, and pruning methods.

2.8.5 On Growing Intelligence

For Jari Vaario and Setsuo Ohsuga (for more details see also [431]), intelligence is the capability of adaptation to a given environment. They identify the following forms of adaptation [433]:

- development (=ontogeny);
- learning (=epigenesis);
- natural selection and genetic changes (=phylogeny).

In their "On Growing Intelligence" paper, the above mentioned forms of adaptation are simulated in the context of autonomous systems. Designing an autonomous system for a dynamical environment by hand is a difficult task since it is almost impossible to foresee all possible situations and behaviors. Nature's answer to this problem is adaptation. Vaario and Ohsuga illustrate the life cycle of a nervous system as shown in Figure 2.14.

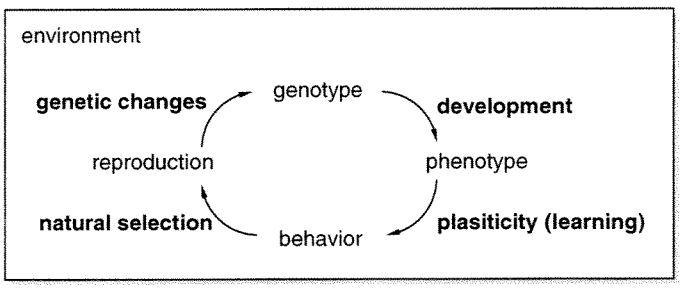

Figure 2.14: The life cycle of a nervous system according to Vaario and Ohsuga illustrating the different forms of adaptation: development, learning, natural selection, and genetic changes.

Vaario and Ohsuga's overall model consists of three interleaved models: (1) neurons, (2) organisms, and the (3) environment. The modeling method used is based on the production rules resembling L-systems [239]. *Interleaved L-systems* are L-systems that may have other L-systems inside it and that are able to modify these systems as well. "For example, the L-system of the environment has several organisms which are defined by L-systems" [433]. The developmental process (the L-system) grows the neural structure, whereas the evolutionary process gradually generates more complex systems. Vaario and Ohsuga successfully applied his method to the

design of Braitenberg vehicles [49]. The biggest restriction of the model and its implementation was that large scale simulations with many individuals were almost impossible due to the computational resources required. Another major problem was the system's bootstrapping: it was extremely difficult to find initial production rules that allowed for further and more complex development. Thus, they explicitly designed some simple systems first and started evolving them to more complex ones. However, this still was no guarantee that more complex systems evolved. In the future, they want to include neural plasticity based learning for the organisms.

2.8.6 Cells that Divide, Differentiate, and Die

Another model that uses artificial genetic regulatory networks to control the development of artificial neural networks has been proposed in 1997 by Peter Eggenberger [119]. The model is particularly interesting because it evolves three dimensional neural networks by means of strictly local interactions between the cells and the genes only. Each cell basically contains the same genome and cell differentiation is achieved due to different signals that each cell gets from the environment, i.e., from other cells. The genetic regulatory network controls cell division, cell differentiation, and cell death. The model also includes cell migration. Whereas the number of genes in the genome of most models proposed so far grew with the number of neurons, the genome in Eggenberger's model does not necessarily grow with a growing number of neurons. This might be crucial for an efficient implementation of large networks. For gene regulation and expression and for cell differentiation, a sort of artificial chemistry has been implemented. As the cells become different, they will produce different substances. Two different classes of cell adhesion molecules are used to connect two neurons together. The connection further contains a weight that can be changed by a Hebbian rule and synapses are either inhibitory or excitatory.

Although the model seems not having been applied to real-world applications, it is a promising approach. The construction of neural networks was possible with no explicit encoding of the network structure, the cell types, the cell's position, and interconnections. In the future, Eggenberger wants to automatically build re-entry maps between different neural network layers in order to obtain networks that can automatically adapt to different tasks.

2.8.7 Phenotypic Plasticity

In [299], Nolfi and Parisi present simulations of the evolution of populations of neural networks. They principally had two objectives:

1. to purpose a more realistic genetic coding of neural network architectures that better approximates biological facts, and

2. to study the genotype-to-phenotype mapping as a process with a temporal notation.

So far, most work that applied genetic algorithms to the construction of artificial neural networks uses an instantaneous genotype-to-phenotype mapping, i.e., organism are born with a fully constructed ("mature") nervous system. The network can the only be changed by applying some form of learning. Nolfi and Parisi note that "[i]n real organisms, however, the mapping from genotype to phenotype is not instantaneous but takes time extending from conception well into extra-uterine life." Nolfi and Parisi's method allows to establish connections during the growth process. The connection between two neurons is established when a growing axonal branch reaches the neuron. For testing their animats, they used a two-dimensional environment divided into square cells. Each animat is equipped with sensors that allow to detect the food elements that are randomly distributed in the environment. At any particular moment, an animat occupies one single cell and it can only move one single at each moment. When the animat senses a food pellet, it will eat it and the food disappears. The simulation results have shown that evolution chooses very simple network architectures in terms of the number of units and connections used. The architectures also tend to be structured in functional sub-nets. Furthermore, they think that by integrating a growth process during the life of an individual organism, "[...] the way is open to study genetically controlled neural and behavioral development as an additional dimension of change besides evolution and learning."

In an extension of their work, Nolfi *et al.* [298] allowed a neuron to grow its branching axon only if the neuron's activation variability exceeds a genetically specified threshold. Thus, the environment influences the neural development, in other words, the genotype is environmentally sensitive and develops into a different phenotype as a function of the environment. This is also called *phenotypic plasticity* (for more details see [356]). The extension of their work might be considered as a simulation of growth factors as they exist in nature.

2.8.8 Artifical Intellects

Visionary brain builder Hugo de Garis has two goals in his life[3]: (1) to build artificial brains, and (2) to raise the alarm on a possible gigadeath artilect war. His interest for genetic programming, artificial neural networks, and artificial embryology goes back up to his Ph.D. thesis in 1992 [97]. Since his thesis, de Garis concentrated on the so called CAM-brain machine (CAM stands for Cellular Automata Machine). The CAM-brain machine is a highly

[3]Source: `http://www.cs.usu.edu/~degaris/`.

specialized FPGA (Field Programmable Gate Array) based machine that allows to grow and evolve three-dimensional cellular automata based artificial neural networks. De Garis approach for building artificial brains is principally based on the number of neurons only. He believes that the more neurons a system has, the more intelligent it will become, however, without giving any definition of "intelligence" (if there exists one). As Howard Gardner says, "[b]efore intelligence can be enhanced or artificially created, it has to be defined" [156]. The CAM-brain machine is certainly impressive, but brute computational power should definitely not be confound with "intelligence". For more details on the CAM-brain hardware, see for example [100, 222]. Despite his rather controversial approach, the model he uses is very interesting as it combines growth, evolution, and neural networks in three-dimensions. The CAM-brain machine can evolve neural modules—each consisting of up to $1'152$ neurons—with a highly complex an almost arbitrarily interconnection topology in three dimensions. A neural module can receive signals form up to 188 other brain modules and can send signals to up to $64'640$ other modules. Axons and dendrites are capable of multiple branching, forming hundreds of connections within each module. The dentritic and axonal structure is directly evolved in hardware using genetic algorithms. Note that, in order to evolve millions of neurons in the CAM-brain, the FPGAs are time shared between multiple neural modules during an fitness evaluation run. The neural model is called *CoDi* for "Collect" and "Distribute" [99, 223]. Figure 2.15 shows the three different cell types: neuron, dendrite, and axon cells. Each neuron can have up to six axons and up to six dendrites. Note that if there are N axons, there will be $N - 6$ dendrites. A 4-bit accumulator in each neuron sums up incoming signals and fires when a threshold is exceeded. Inputs can further be configured (determined by the neuron's chromosome) as inhibitory or excitatory and thus either adds or subtracts from the internal accumulator. A dendrite cell can have up to five inputs (and one output) that are fed into a logical gate (e.g., OR or XOR gate). Likewise, an axon cell can have up to five outputs (and one input). During evolution, blank cells perform no function and are used to grow new dendrites and axons. The growth process starts form a blank cellular space with some neurons determined by genetic algorithm. The growth of axons and dendrites then alternates with each clock cycle. Growth directions are guided by the 6-bit chromosome (one for each direction) assigned to each cell. A blank cell that receives a growth signal from one of its neighbor cells becomes for example a dendrite cell or an axon cell. This rather simple mechanism allows to grow very complex three-dimensional networks. After the neural growth phase, the network is switched to the signaling mode.

The CoDi model works with streams of 1's and 0's. The interpretation is thus not straightforward and a more sophisticated coding, both efficient and evolvable might be useful. The new coding chosen is called *Spike Interval*

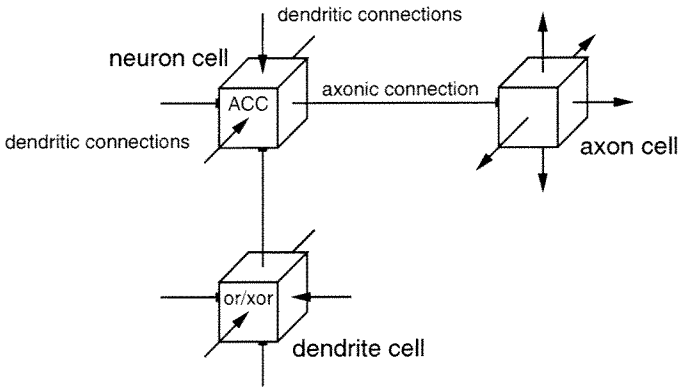

Figure 2.15: CAM-brain 3D cellular automata space with neuron, dendrite, and axon cells. Source [223].

Information Coding (SIIC). The SIIC representation—a sequence of bits or spikes—is inspired by Rieke *et al.* [340]. The procedure for decoding such a spike train consists of convolving it with a special *convolution filter.* The result obtained is called the *estimated signal,* a time-dependent signal that is output from the neural network module. The inverse process, namely, an algorithm which takes as input a binary numbered time-dependent ("analog") signal and outputs a spike train, uses a *deconvolution filter.* The SIIC representation thus requires additional convolution and deconvolution filters. For more details about SIIC, see for example [223].

Initial evolution results using the real CAM-brain machine (and not just simulations) are reported in [98]. The experimental results are rather modest and the initial goal of controlling a kitten robot is by far not (yet?) reached. Nevertheless, work on the next generation CAM-brain (BM2) has already started [101].

2.8.9 Wrap-Up: Where to Go?

The main conclusion of this section is basically the following: attempts to build POE machines and models have been made well before the introduction of the POE model. In my opinion, there is little interest and need for pure POE systems and hardware. Why this? I believe that engineering together three independent representatives — one for each axis — of the POE model (such as a combination of evolutionary artificial neural networks with a developmental genotype-to-phenotype mapping) is probably the most artificial and unnatural way to obtain an "intelligent" adaptive system. The problem lies usually not in the sole fact that learning, evolution, and development are combined together, but that the three forms of adaptation are applied one after another and are not integrated seamlessly and inseparable as in

Nature. This can be illustrated by means of classical reaction-diffusion (RD) systems, which cannot be easily classified by the POE model. RD systems can model evolutionary phenomena such as population dynamics and they develop and adapt through the interaction with the environment. The three axes can be observed, but they are part of all and the same model, which is of course highly desirable. In Bertram and Mikhailov [42], for example, Figure 12 very nicely illustrates the splitting of a cell in their reaction-diffusion system. One could image that the cell contains traveling or standing waves which represent a certain amount of information. This information would then be replicated during cell division, similar to what happens when a natural cell divides. Traveling waves in reaction-diffusion systems can also be used to simulate logical gates (see for example [189,391,392]), which makes such systems universal from the computational point of view.

My personal suggestion is, that instead of artificial POE systems, complex dynamical — and preferably nonlinear — systems should be used which inherently *contain* the three forms of adaptation already, without having engineered them together. A step in the right direction has for example been made by Miller *et al.* [268] and also by Mead [263] some time ago already, who propose to directly exploit the characteristics of the underlying physics (for evolutionary processes, in the case of Miller *et al.*). Another recent step, allowing to bridge in some way living and nonliving matter, comes from Rasmussen *et al.* [334], who assembled non-biological materials into a proto-organism. And there is of course also the possibility to go beyond silicon and to use novel materials, nanotechnology (see for example [189, 256, 384] or [259] for a rather non-scientific introduction), or also chemical-based computing [189,330].

A further (and still personal) consideration is the following: as long as we work with (discrete and mostly single processor) systems and computers as we know them, biological inspiration seems of a rather limited interest. The reason is that information processing in Nature and in computers is fundamentally different [76,77,473]. Computers need a different approach and "biological inspiration" with computers is, overstated of course (!), similar to trying to let a train float as a ship. As we have seen on page 2, modern airplanes don't flap wings either. On the other hand, I am convinced that natural computation has its full right to exist in more natural systems, i.e., "computational" substrates that are more close to physics and biology. I am thinking for example about organic systems coupled with silicon, e.g., neurons growing on silicon, fine-grained massive parallel machines such as CAs and particle swarms, amorphous computers, nanotechnology, etc.

Finally, I guess I have to make clear that I am not at all an opponent of natural computation, on the contrary (otherwise I would not have finished this thesis). However, I am convinced — and this thesis helped me very much to come to that conviction — that a *very* differentiated view and approach is required and that a blind belief in biologically-inspired machines will end

up as failure. Systems and machines inspired by Nature are not blindly to be put on a par with "intelligent machines", as only too many researchers and especially journalists like to do.

In the next few sections, I'm going to present three somehow even more unconventional and biologically-inspired methods of computation.

2.9 Amorphous Computing

In a 1996 white paper[4], Abelson *et al.* first described the philosophy of *Amorphous Computing (AC)*[5] [2, 282]. Amorphous computing is the development of organizational principles and programming languages for obtaining coherent global behavior from the local cooperation of myriads of unreliable parts—all basically containing the same program—that are interconnected in unknown, irregular, and time-varying ways (Figure 2.16). In biology, this question has been recognized as fundamental in the context of animals (such as ants, bees, etc.) that cooperate and form organizations. Amorphous computing brings the question "down" to computing science and engineering. Using the metaphor of biology, the cells cooperate to form a multicellular organism (also called *programmable multitude*) under the direction of a genetic program shared by the members of the colony. Figure 2.17 shows three typical amorphous computing applications: (1) distributing autonomous sensors in a natural environment, (2) painting amorphous particles on a surface, and (3) novel integrated circuits (silicon or not) based on billions of randomly arranged particles.

An exciting new field for amorphous computing might be new printing techniques which allow to create cheap, flexible sheets of transistors — a process that could radically change the way electronic circuits and for example flat-panel screens are created[6]. Using electricity-conducting ink that bonds to a flexible plastic sheet, the goal is to create arrays of bendable semiconductors that work much like their rigid counterparts. Currently, the printable chips are usually not nearly as efficient and reliable as their silicon counterparts, but that is exactly where new computer architectures and organizing principles might be applied.

Amorphous computing us also closely related a field know as *perceptual and sensory augmented computing.* Perceptual computing is concerned with

[4] `http://www.swiss.ai.mit.edu/projects/amorphous/white-paper/amorph-new/amorph-new.html`

[5] AC website: `http://www.swiss.ai.mit.edu/projects/amorphous/`. Note also the nice symmetry between AC (amorphous computer) and CA (cellular automata), which are complementary in many aspects! I'd prefer to use AComp instead of AC because of the possible confusion with artificial chemistries (AC), however, AC is more commonly used in the literature.

[6] The Xerox Palo Alto Research Center (`http://www.parc.com`), for example, is working on such printing techniques.

the processing of sensor data such as audio, video, and other sensory data. The growing interest and the increase in the amount of sensor data calls for efficient techniques to index, search, and structure those data. In the future, the number of sensors may increase substantially since many types of sensors become very cheap such that computing has access to and can be enhanced by ubiquitous sensors. Ubiquitous sensors and sensing have the potential to fundamentally change the way we think about human-computer interaction and how machines perceive the world.

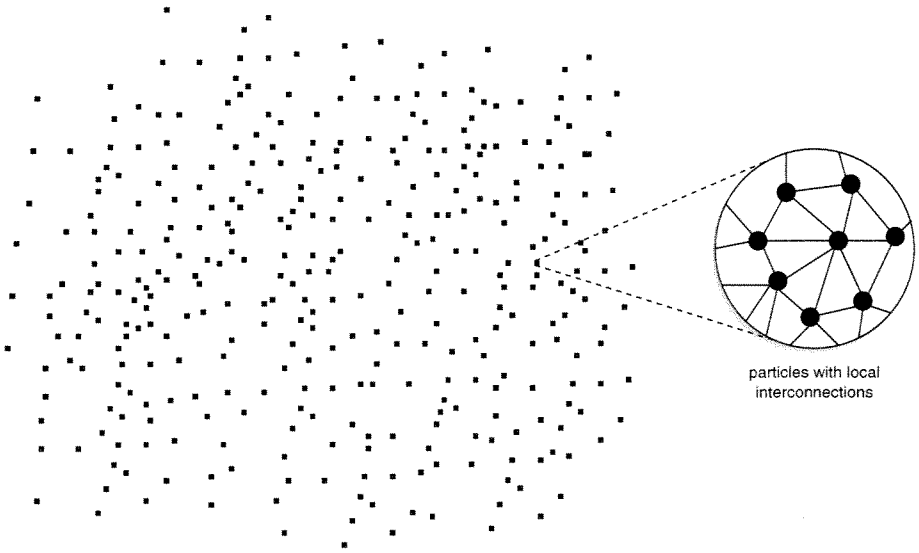

Figure 2.16: An amorphous computer particle swarm.

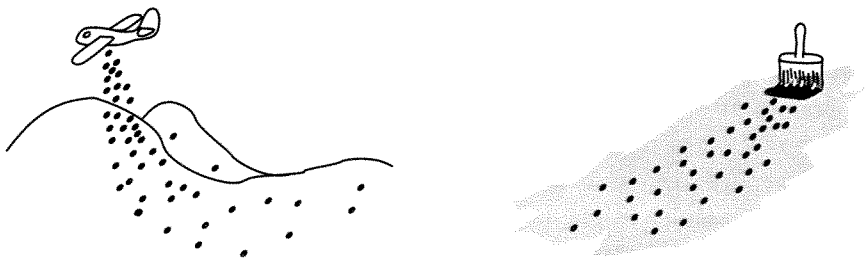

Figure 2.17: Typical amorphous computing applications: (1) distributing autonomous sensors in a natural environment, and (2) painting amorphous particles on a surface.

The goal of the amorphous computing research is to create the system-architectural, algorithmic, and technological foundations for exploiting programmable materials (e.g., a paintable computer [57]) that react to each

other and to their environment. Abelson *et al.* present their vision as the following fundamental challenge:

> "To obtain desired, coherent behavior from large numbers of unreliable parts that are interconnected in unknown, and time-varying ways, by deliberately orchestrating their individual behavior and their cooperation."

Amorphous computing is inspired by the recent developments in biology (understanding the genome, the developmental processes, etc.) and in micro fabrication (new materials, nanotechnology, etc.) that will make it possible to build or grow huge numbers of almost identical information processing devices at almost no cost. Whereas existing technologies such as VLSI, FPGAs, printed circuits, etc. allow to arrange components in a pre-specified manner, it will be crucial for novel smart materials to obtain a coherent global behavior without the need to precisely arrange and interconnect components. What researchers wish to engineer does biology already very well, namely the dynamic organization of cells into highly complex and differentiated organs. It is not the goal of the original amorphous computing project to study and use the principles of self-organization *per se*, but rather to construct and engineer systems that do what they should do. Compared to the traditional programming of massively parallel systems, amorphous computing presents a even greater challenge as its mechanisms must be independent of the arrangement, interconnection, and reliability of its elements.

2.9.1 Definitions

The basic model is rather simple and consists of numerous identical elements (or particles) that are randomly arranged on a surface or in a volume (Figures 2.16 and 2.17). Note that most current amorphous computing projects are limited to a two-dimensional surface. It is usually also assumed that the particle density is sufficiently high so that all processors are connected and that the variance in density is low. Each particle has modest resources and computing power available only since they have to be as cheap as possible. Each particle can directly communicate only with a limited number of neighbors (see Figure 2.18). In the original model, all processors share the same channel (i.e., wireless broadcast), which means that collisions can occur when two processors within the same communication radius send out a message simultaneously. A collision might be detected by the receiver (garbled message). The basic paradigms of amorphous computing might seem similar to cellular automata at a first glance, however, each particle of an amorphous computer can in principle communicate with a different number of neighbors and the neighborhoods can overlap. Furthermore, the communication channels as wells as the particles are not reliable. Fault tolerance

in an amorphous computer is basically achieved via redundancy in hardware and via redundancy on the representations rather than via hardware perfection.

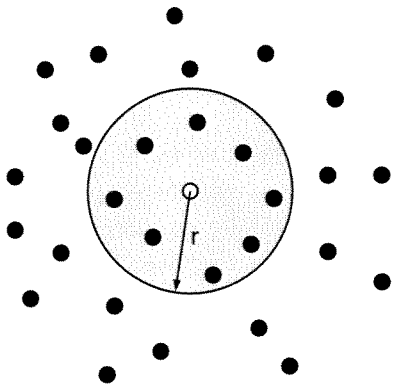

Figure 2.18: The communication model of an amorphous computer. Each particle can communicate with its neighbors within a communication radius r that can be different from one particle to another. All particles share the same communication channel and collisions can therefore occur.

The particles do not possess a knowledge of their location and orientation within the amorphous computer and there is no global synchronization signal available. The basic concept of amorphous computing allows the particles to become mobile, however, to the best of my knowledge, all work in the field assumed fixed positions. Moving particles presents an additional challenge which has addressed especially in mobile networks already. Routing in large-scale mobile ad hoc networks[7], for example, is challenging because all the nodes are potentially moving. Geographic routing can partially alleviate this problem, as nodes can make local routing decisions based solely on the destinations' geographic coordinates. However, geographic routing still requires an efficient *location service*, i.e., a distributed database recording the location of every destination node. Since amorphous particles are basically very simple, it is almost impossible or at least difficult to implement distributed databases that manage myriads of elements. This represents not a fundamental problem, but rather a technical challenge related to the limited number of resources available and to the large number of nodes.

The amorphous computing properties can be summarized under the following assumptions (see also [81]):

- Individual processors are identical and mass produced.
- Processors possess no *a priori* knowledge of their location, orientation, or neighbors' identities.

[7] See for example `http://www.terminodes.com`.

- Processors operate asynchronously although they have similar clock speeds.
- Processors are distributed densely and randomly.
- Processors are unreliable.
- Processors communicate only locally and do not have a precision interconnect.
- The amorphous computer can exist on a 2D surface or in a 3D space. It is assumed that the number of particles is very large.
- The communication model assumes that all processors have a circular broadcast of approximately the same fixed radius (large compared to the fixed size of a processor) and share a single channel.

Formally speaking, an amorphous computer can be defined as in Definition 2.9.1 (see also [284]).

Definition 2.9.1 (Amorphous Computer)
An amorphous computer can be defined as a tuple

$$\mathsf{AMC} = (V, E) \tag{2.3}$$

that represents a directed graph and where:

1. $V = \{(x_1, y_1, z_1, c_1), \ldots (x_N, y_N, z_N, c_N)\}$ *is the set of particles, i.e, the nodes of the graph,* (x_i, y_i, z_i) *is the physical position of the particle* i *within the amorphous computer related to some coordinate system,* $C_{min} < c_i < C_{max}$ *is the communication radius of particle* i*,* C_{min} *and* C_{max} *guarantee that* c_i *is of approximatively the same value;*
2. $N = |V|$ *is the number of particles;*
3. $E = \{(i,j)|i,j \in V \wedge r = \sqrt{x^2 + y^2 + z^2} < c_i\}$ *is the set of edges,* r *is the physical distance between the two nodes* i *and* j*;*
4. $n(i) = \{j|(i,j) \in E\}$ *is the local neighborhood of processor* i*;*
5. $d(i) = |n(i)|$ *is the degree of the neighborhood of processor* i*;*
6. $d_{avg} = \frac{1}{N} \sum_{i=1}^{N} d(i) \ll N$ *is the average neighborhood size;*

Each processor occupies a physical space and they cannot be superposed. Therefore, a limited number of processors can be placed on a given surface and within a communication region. The following equation holds:

$$d(i) < \rho_{max} = \frac{\Pi r^2}{processor\ size} \tag{2.4}$$

Furthermore, the number of neighbors within h hops of a processor is physically upper bounded by $h^2\rho_{max}$. Figure 2.19 illustrates how the processor arrangement of an amorphous computer can be represented in the form of a directed graph. The graph is directed as communications may not be bidirectional.

■

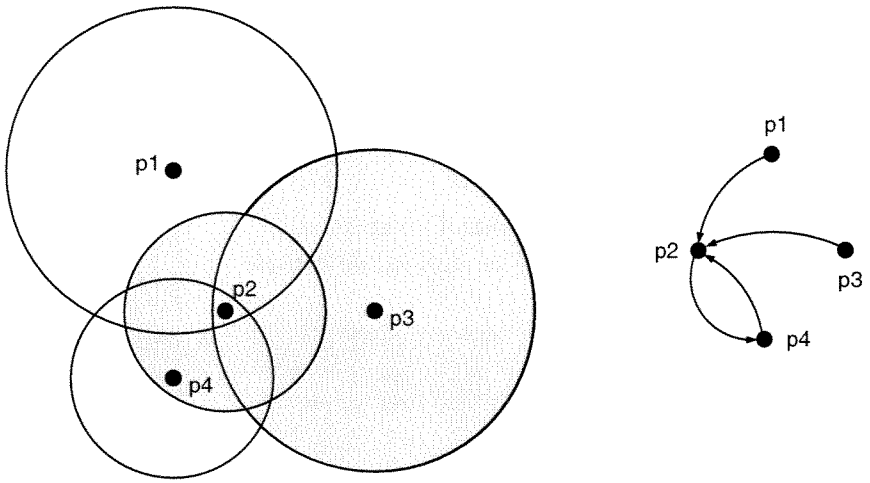

Figure 2.19: An amorphous computer with its communication model can be represented in the form of a graph. On the left, four processors with a different communication radius r. On the right, the directed graph. Nodes represent processors, edges communication channels.

Alternatively, the interconnection topology of an amorphous computer might also be defined by means of the concept of the *NK* network model (with adjacent neighborhood) as presented in Section 3.6.

Definition 2.9.2 (*NK* Network Amorphous Computer Model) *According to Definition 2.9.1, an amorphous computer might also be considered as an* NK *model where:*

1. $N = |V|$ *is the number of particles; and*

2. $K = d_{avg} = \frac{1}{N}\sum_{i=1}^{N} d(i) \ll N$ *is the average neighborhood size;*

■

Kauffman's findings on RBNs do naturally, as its name says, only apply to randomly constructed boolean networks where the transfer function of each node is specified at random too. It would therefore be wrong to generalize the results of Section 3.6 to amorphous computers since their particle function is not usually specified at random.

2.9.2 Global Behavior from Local Interactions

Decades of research have failed to produce robust, general methods for programming massively parallel systems. The main and fundamental challenge is to obtain a global behavior from local interactions only. One of the most explicit examples are certainly cellular automata. Typical applications are the synchronization and the density task (see Section 3.5.1 or [60,411] for an overview) for cellular automata. There are a lot of similarities and identical fundamental problems between programming a cellular automata, an agent-based system, a massively parallel computer, and an amorphous computer. Ultimately, programming such a system comes down to specify *microscopic* interactions to attain predetermined *macroscopic* goals. This is definitely a non-trivial problem since the local-to-global relationship is all but well understood and there exists no generally applicable method or theory. In the synchronization task, the global goal is to synchronize all cells, the local interactions are the rules of the CA's cells. Finding these rules can either me done by hand, which becomes rapidly impossible with larger CAs, or by making use of an optimization technique such as evolutionary algorithms. In general, amorphous computing presents a greater challenge than cellular automata because its mechanisms must be independent of the detailed configuration and reliability of the individual particles.

In the following, a overview on some of the metaphors from biology applied to amorphous computing shall be described. I will mainly follow Ellie D'Hondt [108], who provides a comprehensive overview on the topic.

Forming Groups, and Hierarchies

Hierarchical, structured and grouped organization in biology and Nature is very common. On the one hand, an aggregation of individual elements into groups is useful to optimize limited resources but also to fault-tolerance. On the other hand, it is also a mean to reduce complexity to minimize the overhead for control.

Many different papers have suggested aggregation and hierarchies of aggregates as a possible mechanism for programming and hiding complexity

(see also Section 3.2.1). *Concurrent Aggregates* [65], for example, present a dynamically-typed concurrent object-oriented language based on aggregates that are being treated as objects which are to be used to program a parallel machine. The principal difference between such rather "classical" approaches and hierarchies on an amorphous computer—as we will see—is that there exists a very close relation between the amorphous computer spatial structure and the distribution of the tasks.

Not unlike a hierarchical organization in biology, an amorphous computer can also be organized in hierarchical groups that act as single entities on a higher level of abstraction. The 1998 MIT AI memo [284] describes an efficient algorithm, called *clubs*, that allows to organize an amorphous computer into groups by making use of the local communication. Time taken is proportional to the local density of processors, even in an asynchronous setting. Furthermore, the algorithm does not need to have access to global information on the position or IDs of the particles nor to the topology of the amorphous computer. Traditional group forming algorithms for point-to-point networks are either difficult to implement without a huge loss of efficiency or they require synchronizers which in turn generate large numbers of messages.

A *club* is a group of processors that satisfies the following main properties:

1. all processors should belong to some group;
2. all groups should have the same maximum diameter; and
3. a group should allow for local routing, i.e., all processors within a group should be able to communicate with each other by using only processors within the same group.

Algorithm 2 presents the club algorithm that allows to form groups with a maximum diameter of two hops. Each club has a leader that is chosen in the following way: each processing elements has a random number generator that initially generates a random number from the fixed interval $[0, R[$. The processors decrement their random number silently until they are being interrupted by a message from one of their neighbors or until they reach 0. If a processor reaches 0, it declares itself a leader and recruits all its members by sending them a "recruit" message. The dimension of a group is therefore determined basically by the communication radius. If a processor is declared a follower (by receiving a message before reaching 0), it stops counting and waits for a "recruit" message. A processor can therefore also belong to several groups, i.e., a processor can be follower of several leaders. If a collision is detected by a processor, it assumes that more than one of its neighbors tried to recruit it a the same time and it becomes a follower and will figure out its leader later. The algorithm completes when all processors

are members of some group. To remove leadership conflicts, the algorithm can also be run several times.

Figure 2.20 shows leaders (= dark spots) that form groups. The circles represent the local broadcast region. All processors within this area are followers of the leader.

A number of interesting properties, theorems, and results around the *clubs* algorithm are presented in [284]. The interested reader is referred to this paper (which is also available on the MIT's amorphous computing website).

Algorithm 2 *Overlapping Clubs* Algorithm. Source: [284]

```
1: integer R = upper bound for random numbers
2: boolean leader, follower = false
3: t_i = R
4: r_i = random[0, R[
5: while not(follower) and not(leader) do
6:    t_i = t_i − 1
7:    if r_i > 0 then
8:       r_i = r_i − 1
9:       if not_empty(msg_queue) then
10:         if first(msg_queue = "recruit") then
11:            follower = true
12:         end if
13:      else
14:         leader = true
15:         broadcast("recruit")
16:      end if
17:   end if
18: end while
19: while t_i > 0 do
20:    listen for other leaders
21:    t_i = t_i − 1
22: end while
```

In another MIT AI memo [81], Coore, Nagpal, and Weiss present an approach to programming an (unstructured) amorphous computer by spontaneously organizing an unstructured collection of processing elements into cooperative groups and hierarchies. This paper introduces a structure called an *AC Hierarchy*, which logically organizes processors into groups at different levels of granularity. The AC hierarchy simplifies programming of an amorphous computer through new language abstractions, facilitates the design of efficient and robust algorithms, and simplifies the analysis of their performance. The AC hierarchy is an ordered sequence of levels that can

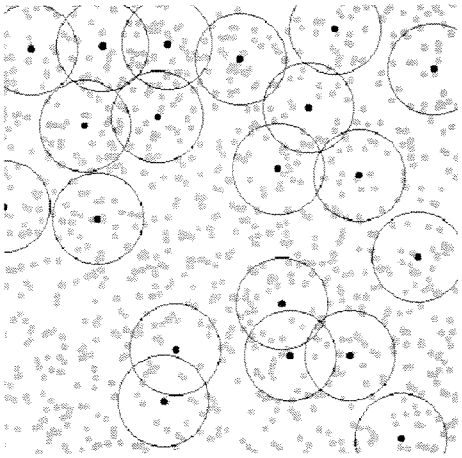

Figure 2.20: Leaders (= dark spots) that form groups. The circles represent the local broadcast region. All processors within this area are followers of the leader. Source: [284].

be considered in a graph representation. In a graph of level n, a group is a node and an edge represents communications between adjacent groups. A group at level n is a connected component of the level $n-1$ graph. Figure 2.21 shows a possible AC hierarchy. The 1997 memo also describes several example applications that make use of the hierarchical structure. For example, the *mergesort* algorithms makes us of the hierarchies to distribute the problem and to combine the answers [284, p. 7]. Another field of interest is the distributed control and collection of sensory input data. The problem of hierarchical control—where decisions are relied on a subset of the information available—is not well understood in general.

In [284], two algorithms that build first level structures are presented as well as an algorithm that constructs higher levels of the hierarchy. The above presented overlapping *clubs* algorithm 2 allows to build a first level organization based on groups with leaders. A second algorithm, the *tight-clubs* [284, p. 16ff] algorithm, addresses the issue of fault-tolerance and constructs clubs that are more tightly coupled. A failure of the leader in overlapping clubs generated by algorithm 2 generally results in a failure of the entire group because its members become potentially disconnected. The *tight-clubs* algorithm simply produces groups of processors where all members of a given group can communicate with each other directly. It follows that any member can be a leader and that the failure of individual members does not affect the connectivity within a club.

In order to generated higher levels of the hierarchy from the first level structures, the so-called *tree regions* algorithm might be used. This algo-

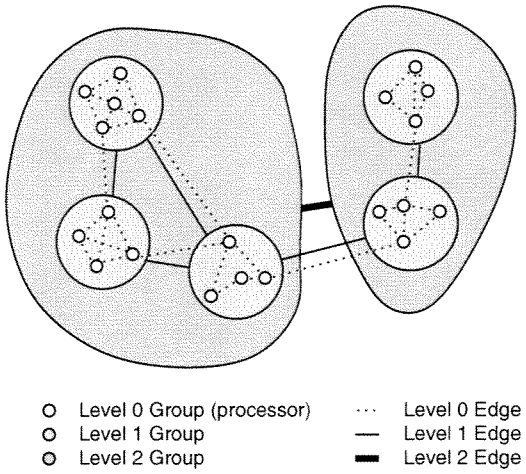

Figure 2.21: An amorphous computer hierarchy. Redrawn from: [81].

rithm takes advantage of the coordinated group behavior and inter-group communication capabilities that are provided by the immediately lower level to construct processor groupings. The algorithm can be applied recursively to create multiple levels. The construction of level n groups starts with a leader selection algorithm at level $n-1$ alike to *clubs* algorithm presented above. The main difference is that an extra parameter h is introduced to determine the diameter bound of the groups formed. Once a group is elected leader, it simply becomes a tree root and seeds a tree of fixed height h by recruiting is neighboring groups as children. The child group then tries to recruit is own neighbors, unless it is itself at depth h from the root. The complete algorithm is reprinted in [284]. Figure 2.22 shows the early stage of trees sprouting on top of overlapping clubs.

In summary, the presented methods allow to self-organize an amorphous computer into groups and hierarchical organizations. Several applications have been presented in [81, 284], however, they remain principally in the domain of toy applications (which is just fine as long as the the research is still fundamental).

Establishing Coordinate Systems

The particles of an amorphous computer do not have *a priori* knowledge on their physical position and orientation. For numerous applications (e.g., interpreting sensor data, intelligent environments, etc.), however, it might be necessary that the processing elements know their position within the amorphous computer, i.e., related to the other elements.

Two approaches have been proposed by the amorphous computing group:

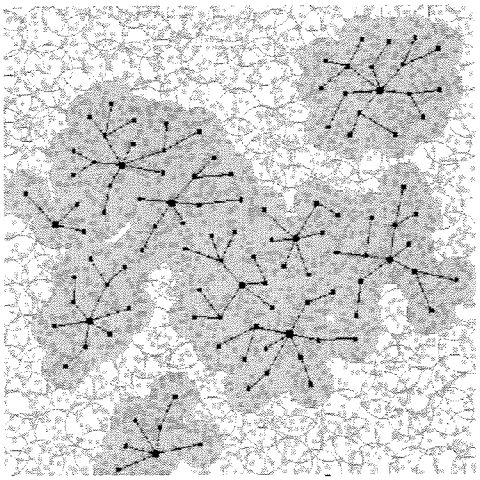

Figure 2.22: Early stage of trees sprouting on top of overlapping clubs. Source: [284].

1. Radhika Nagpal [281] suggest an algorithm that is inspired by biology and that makes use of chemical gradients and

2. Daniel Coore's method [79] generates established a coordinate system by solving Laplace's equations within a circle.

The first approach (see [281]) is inspired by developmental biology. Cells with identical programs (i.e., DNA) use many different techniques to robustly determine their position relative to other cells and within an organ or organism. According to Wolpert, position information is crucial for pattern formation [466]. Nagpal uses an algorithm that is inspired by the development of the drosophila embryo and which is based on chemical gradients. The basic idea is the well-known fact that a point in a two dimensional plane can be uniquely relative to three non-colinear reference points. The algorithm can be summarized as follows:

1. First, three non-coliner refrenece points (anchors) are chosen, either by an external agent or by a leader election algorithm.

2. Then, each anchor produces a unique gradient that allows the other processors to determine their distance from the three anchors.

3. Finally, a triangulation formula is used to convert the distances to cartesian coordinates relative to the anchors.

The gradient algorithm of step 2 is rather simple and can be implemented straightforwardly. Each anchors sends out messages to all of its neighbors

with a count set to one. All neighbors forward these messages and thereby increment the count by one. Each processor stores the minimum counter value received and ignores messages with higher values. Hence, a wave of wave of messages propagates forward from each anchor.

To improve the accuracy, Nagpal also proposes a smoothing algorithm that allows to obtain a better resolution. The following three main sources of errors in the position estimation algorithm can be identified:

1. errors in the gradient step due to the discrete distribution of processors;
2. errors in the smoothing step due to variations in the processor densities; and
3. region specific errors in the triangulation step.

A detailed theoretical analysis presented in [281] has revealed that there is a critical minimum average neighborhood size of 15 processors to obtain a good accuracy. Furthermore, there also exists a strict fundamental limit on the resolution of any coordinate system that is established from local communications only. Interestingly, randomly distributed processors also yield in higher precision than regularly arranged grids. The interested reader is referred to [281] for more details.

The second approach (see [79]) establishes a coordinate system on an amorphous computer by solving differential equations. The coordinate approximation is based on solving the partial differential equation $\frac{\delta u}{\delta t} = \nabla^2 u$ where u represents the approximation to either x or y (two equations are being solved simultaneously). The method only works when the origin of the coordinate system is specified either by an external supervisor or by randomly selecting a processor. The algorithm seems to produce similar average errors as Nagpal's approach, but is more sensitive to variations in density as it is based on an iterative averaging of neighbor information. In addition, it difficult to detect the algorithm's termination locally.

2.9.3 Programming Amorphous Computers

Probably the first serious attempt to develop engineering principles and languages to program an amorphous computer was the *growing point language (GPL)* [80] that allowed to express complex patterns. The key element of GPL is the botanical metaphor of the *growing point*, a locus of activity that essentially describes a path through the connected elements in the system. A growing point is an activity of a group of neighboring particles that can be propagated to an overlapping neighborhood. Growing points can split, die off, or merge with other growing points. At each moment in time, a growing point only resides in a single location, called the *active site*, in the GPL domain. A growing point propagates by transferring its activity from a particle

to one of its neighbors according to its *tropism*. At its active site, a growing point can deposit *material* and it may secrete *pheromones*. A growing point's tropism is specified in term of the neighboring pheromone concentrations that determine a scalar field whose gradient directs the motion of the growing point. The path of a growing point, however, is only detectable it material—represented by a different color—was deposited (not unlike the pheromone trails of ants). Daniel Coore has demonstrated that any prespecified planar graph can be generated—up to connection topology—by an amorphous computer under the control of a growing point program, provided the distribution of particles is sufficiently dense.

I will not go into further details. The interested reader is invited to read the comprehensive overview provided in [108] (or also [2]) or the original work by Coore [80].

Another model [456] has been proposed by Weiss that allows to program the particles by means of a very simple language. The program to be executed by each particle is constructed by a set of independent rules that are enabled by boolean combinations of binary markers included within each particle. The enabled rules can then be triggered by receiving labeled messages from the neighboring particles. Further, rules may set or clear various markers and they can also propagate new messages.

Radhika Nagpal's model [283] is based on a sheet of identically-programmed, flexible, autonomous agents (i.e., cells) that assemble themselves into a predetermined global shape using local interactions only. Using a set of paper-folding axioms ("origami" axioms from origami mathematics), the global shape is described by a folding construction. The language developed by Nagpal is called *Origami Shape Language (OSL)* and allows to compile a global description into the program for each individual particle (see Figure 2.23). This approach differs from approaches based on cellular automata where local rules are constructed by hand, empirically, or by evolutionary approaches. The compilation approach bears other advantages such as the description of the global shape at an abstract level. Each cell basically executes the same program which differ only in a small amount of local information, i.e., the code contains multiple functions that can be activated in response to a number of predefined conditions. The process is best compared to cellular differentiation of real cells. The model is based on a programmable sheet, i.e., the amorphous computer, which has two surfaces: an apical surface and a basal surface. Each cell on the sheet can tell the difference between the two surfaces. If a line of cells all (virtually) fold their apical surfaces, then this causes the sheet to fold until its apical surface touches itself (analogous for the basal surface). The cellular program essentially works based on five primitives (gradients, neighborhood query, polarity inversion, cell-to-cell contact, and flexible folding) that form the basis of how they organize and self-assemble. The origami shape language is then implemented by using these five primitives.

Compiling an origami shape program consists in creating local state variables for each distinct point and line and then translating each origami shape language operation into a call to the corresponding cell procedure with the appropriate arguments.

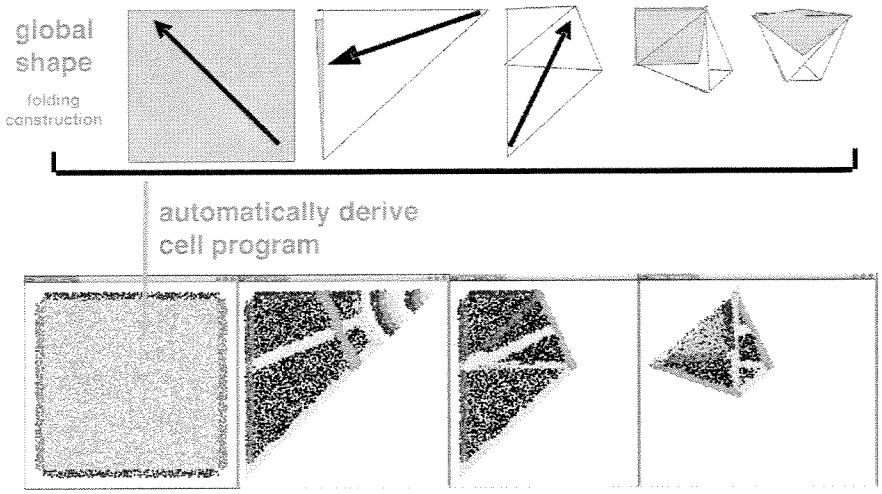

Figure 2.23: The global shape is described by a folding construction on a sheet. The specification is made using the *Origami Shape Language* which is then compiled into the program for an individual cell. Source: [283, p. 10].

Nagpal's approach provides not only a new approach to designing local behavior to obtain a global goal but provides also many insights into the relationship between local interactions and global behavior. A very interesting feature is for example the fact that a cell program specified in the origami shape language is scale independent, i.e., the same global shape is obtained for any size of the cell sheet.

In his recent work, William Butera [57] introduces the notion of a *paintable computer*, a concept that is based on the amorphous computer paradigms.

> "While the details will change en route to practice, this notion of a *paintable* captures the essence of what could be a big part of our computing future: computation as a tangible, fluidly dispersible additive to ordinary objects. Want a surface to be smart? Add a layer of computing. Want it to be smarter? Add a second coat. Has the computer lost its luster? Get out the belt sander" [57, p. 2].

Butera identified the following points as worst hurdles to build a paintable computer:

- **Asynchrony**: no global clock is available.
- **Fault tolerance**: individual parts can fail.
- **Network locality**: particles can only communicate with other particles in the immediate neighborhood.
- **Adaptive topology**: the final topology of a paintable is unknown.
- **Code compactness**: each particle has very limited resources only and communication between the particles is slow
- **Combinatorics**: the unconstrained placement of particles represents a worst case in the combinatorics of the hardware.

From a practical point of view, it is impossible that a human would be able to structure and program a paintable. Butera states: "[...] if we can not get a human to structure procedures, we are going to have to get the procedures to structure themselves" [57, p. 5]. Butera defines a hardware reference platform that is based on conventional and universally programmed modules, however, no real hardware has been built. The programming model of the paintable is based on *process fragments (pfrags)*, autonomous, self-contained executables that migrate among the particles and interact with the local environment. All pfrags running on a particle reside in its RAM space. A part of this RAM is reserved to I/O space, an area which is at least readable by any pfrag running on the particle. Furthermore, a subset of the I/O space is called the *HomePage*. The HomePage is an area where all local pfrags can both read an write (i.e., share) tagged data. This common bulletin-board system has a further important characteristic: when a pfrag posts data in that space, the data will be copied to mirror sites on all the neighboring particles and will become visible there.

Based on the hardware reference platform and the programming model, Butera proposes several interesting applications that he tested on the *Psim* simulator. For example, he showed that it is possible to store packetized audio data in the memory of a particle ensemble. The data is exchanged with the particle ensemble via streaming through arbitrarily positioned I/O ports. The performance of this application was examined on a simulator using a configuration of 660 particles, each communicating with 10 to 16 immediate neighbors.

Another example is the *Holistic Data Storage (HDS)*: given a digitalized image and a memory element embodied as a 2D planar surface, the goal is to select a representation for that image such that it can be stored into the memory plane. If the contents of the memory are read out and decoded, the reconstructed image should be identical to the original. If the majority of the stored data is lost or damaged, the decoded images should still be retrievable in low resolution. In other words, loss of data should only result

in a loss of "sharpness". Experiments were performed on an amorphous computer with 1200 particles and with two color images of 320×240 pixels.

The two other applications presented in [57], a surface bus and image segmentation, shall not be further described here.

2.9.4 Amorphous Hardware

Amorphous hardware is in some way still in a preliminary stadium since it was so far impossible to implement a full-blown amorphous computing media. However, advanced fabrication techniques will allow to synthesize myriads of processing elements so cheaply that they might be delivered in a paint or wrap. It is also believed that the amorphous computing technology will be important to control devices created by nanotechnology. Existing amorphous computing media basically rely on networks of microprocessors. On the other hand, several simulators have been implemented that allow to test and simulate programs and concepts.

The "HC11 GUNK"[8] [108] demo illustrates the first infrastructure for an amorphous computer (see Figure 2.24). As its name suggest, its particles are based on the well-known Motorola HC11 microcontroller. The amorphous particles are communicate by wired broadcast, they control four LEDs, and are equipped with a sensor that can be triggered by, for example, a laser beam. There are a rather modest amount of particles available, but their size does not allow to arrange them randomly and in large number on a surface anyway. Unfortunately, programming the microprocessors is rather complicated and must be done in assembler code.

An interesting application presented on the "HC11 GUNK" is the synchronization task *à la* Mirollo and Strogatz [274] (see also Teuscher and Capcarrère [411]). Their model relies on a monotonically increasing and concave down function which represents the increasing sensitivity of the fireflies to a flash by any other firefly (coupled all-to-all) during the course of one period.

Extending the original model the "HC11 GUNK" demo employs an offset step function to represent sensitivity versus time. Also, the GUNK "fireflies" are coupled some-to-some in contrast to all-to-all. When a GUNK "firefly" hears the flash of its neighbor, it flashes immediately and reduces its own period. When no flash is heard during the course of a period, the "firefly" increases its own period, ultimately eliminating harmonics.

Chapter 3 of [108] provides a detailed presentation of further amorphous computing media and simulators.

[8] http://www.swiss.ai.mit.edu/projects/amorphous/HC11/

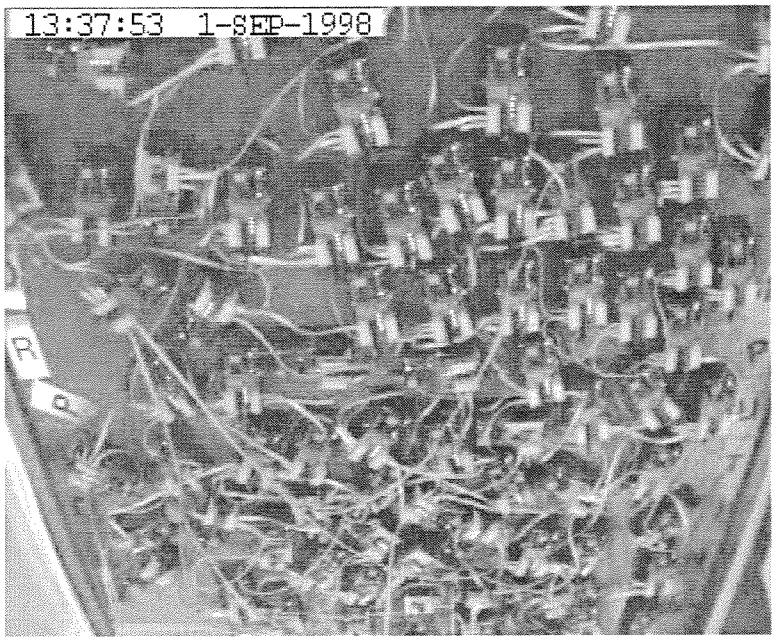

Figure 2.24: The "HC11 GUNK" infrastructure for an amorphous computer based on HC11 microcontrollers. Source: MIT amorphous computing website.

2.9.5 Related Work

Little other work exists in the direction of amorphous computing. Probably the most similar project is ongoing in France at the University of Paris-Sud. The coupled language-machine project—called the *BLOB Computing Project* [172, 173][9]—is aimed at representing an alternative to the well-established von Neumann model. The main goal is to provide a massive parallel and asynchronous computational model that offers a good hardware scalability combined with a language that allows to fully exploit the underlying hardware. The project relies basically on three foundations:

1. a cellular asynchronous language,
2. a fast reconfigurable architecture, and on
3. cellular automata.

The programming language relies on a virtual machine called *Graph Machine* that is a self-modifying and self-developing net of automaton. This language goes back to Gruau's work on cellular encoding (see for example [169, 171]). Each node (=automata) of the graph can locally modify the

[9] `http://blob.lri.fr`

graph and can apply cell division instructions in order to develop the graph. A cell can for example simulate a simplified artificial neuron, thus a cellular code can develop and simulate an artificial neural network. Regarding the underlying massively parallel and fine-grained hardware architecture, the main problem is basically to determine which processor will simulate which cell of the graph. This mapping — which is one of the central problems of parallelism and which is known to be NP-hard — should minimize the distance between the nodes that communicate. Gruau et al. got their inspiration from Nature: the cellular development is strongly influenced by chemical gradients and different physical forces. They used these "placement" principles to map the nodes onto the underlying hardware, i.e., optimizing the shape and the location of a blob is equivalent to solving the mapping problem. Using graphs of blobs on top of a network of nodes also allows to represent various data structures [173] such as finite graphs or hierarchical multisets.

Another project comes from Xerox and is called *Smart Matter*[10]: it is based on massively distributed, low-cost manufactured sensors, actuators, and controllers. The project extends, however, also to material science and includes modular robotics, paper-like displays (e-ink), etc.

With the advent of internet and mobile networks, there was naturally also a growing interest in new algorithms and organizational paradigms. Especially challenging is the field of large-scale ad hoc mobile networks as all the nodes are potentially moving. Many similar problems are encountered as on amorphous computing. Mobile networks work asynchronously and there is usually no *a priori* information available on the position of the nodes. Mobile networks are, however, usually built up from fewer nodes than an amorphous computer and the computational power is much bigger compared to a simple amorphous particle.

Amorphous computing also shares several common points with agent-based systems, however, the question what such an agent really is and how it differs from any other program or unit is, cannot always be clearly answered. Franklin and Graesser [145], for example, present numerous definitions and also provide their own—in an attempt to capture the essence of agency—which shall be given and used here:

> "An **autonomous agent** is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future."

Naturally, agents are also closely related to *collaborative problem solving* (see for example [179]) and *distributed artificial intelligence (DAI)* (see for

[10] `http://www.parc.com/research`

example [255]). The problem of synthesizing and analyzing collective autonomous agents is being studied in the robotics community for some time already. DAI is often divided into two subfields:

- *Distributed Problem Solving (DPS).* DPS deals with centrally designed systems solving global problems and using built-in cooperation strategies.
- *Multi-Agent Systems (MAS).* MAS deals with heterogeneous agents faced with the goal of utility-maximizing coexistence.

Using the above described definitions, amorphous computing might certainly be classified under the notion of an agent system and under the concept of distributed artificial intelligence. Sometimes, amorphous computing is also mentioned under the notion of *swarm computing* [217] to emphasize that a collective result emerges from individual behaviors with the surprising symmetry of a relocating bee or ant colony.

2.9.6 Wrap-Up

Amorphous computing — mainly developed in anticipation of the fast evolving fields of micro-fabrication and cellular engineering — is definitely a highly interesting computing paradigm, that remains, however somehow vague as it its rather an abstract concept than a concrete architecture. But this might of course also be an advantage as it does not impose restrictions. For example, the basic amorphous computing paradigm does neither mention self-organization nor emergence, but of course both can — and even should — be used.

The project makes use of many concepts inspired by biology. For example, the amorphous computing particles are taken as a metaphorical equivalent of cells. On the other hand, biology often makes uses of chemical gradients: an amorphous computing systems can use gradients to establish a coordinate system. One of the main challenges of amorphous computing also consists in identifying the principles that can be borrowed from both physics and biology since both extensively rely on large number of locally interconnected elements, i.e., cells, particles, atoms, etc. Naturally, an amorphous computer is also a massively parallel computer and therefore closely related to cellular automata (see also Section 3.4).

The concept is especially interesting for future technologies and new manufacturing techniques based on a very large number of irregularly arranged and imperfect components. Interactions are local, no global control nor clock is available.

In [176], Haddow and von Remortel put together the technological requirements of several digital evolvable hardware technologies, including

amorphous computers. They summarize, that the technology required to fully exploit evolution would have to be one where complex circuits may be implemented without requiring a large amount of data, fast programmability, and re-programmability. They further mention that amorphous computers are based on biological cellular systems and are thus quite well suited for implementing developmental principles.

2.10 Artificial Chemistries

Chemistry is the branch of physical science that deals with the composition, properties, and reactions of substances. Whereas *organic chemistry* deals with substances that contain carbon, *inorganic chemistry* is concerned with substances that do not contain carbon. Chemistry is both *analytical*, i.e,, substances are being analyzed, as well as *synthetic*, i.e., substances are being created. The beginning of todays chemistry goes back to the 17^{th} century and basically originated in alchemy. However, already the Egyptians and Babylonians possessed some chemical knowledge.

Life itself might be defined as a chemical system capable of self-reproduction and of evolving. It is generally assumed that organisms are alife not because of their properties of their constituents (i.e., chemicals), but rather because of their organization. In contrast to the complex assembly of thousands of organic molecules into a virus, nobody would consider a simple single molecule as "alife".

The field of *artificial chemistries*—which is usually considered as a subfield of artificial life—is mainly motivated by the quest for understanding how life originated and evolved. This attempt is naturally based on the hypothesis (or hope) that by abstracting from the natural molecular processes, the principal characteristics of the "real" chemistry will still be captured.

The field of artificial chemistries deals with abstract (artificial) elements that interact with each other, that might maintain themselves, and that might be able to create new elements. The rather broad concept of artificial chemistries can be applied in many fields such as for example:

- modeling "real" chemical and biological systems;
- computation and unconventional information processing;
- optimization.

Artificial chemistries usually process information in a completely decentralized and massive parallel manner unlike the traditional Neumann computing architecture (see Figure 2.25). It is in general also easy to consider chemistries that are self-timed or operate asynchronously (i.e., without a global synchronization signal).

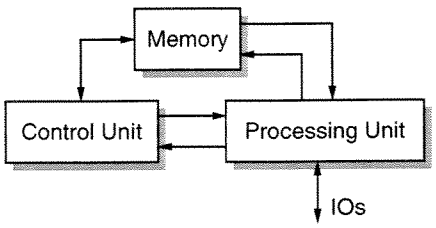

Figure 2.25: Traditional von Neumann computing architecture with a fixed general purpose architecture.

In the following, the basics of artificial chemistries shall be presented and the most important exponents introduced. I will more or less closely follow [112]. For further information, the reader is invited to consult Dittrich *et al.*'s excellent review [112] or his PhD thesis [111] (planned to be published in the form of a book).

2.10.1 Definitions

Adopting the definition of Dittrich *et al.* [112], an artificial chemistry is defined as a triple (S, R, A) where $S = s_1, s_2, \ldots, s_n$ is a set of molecules, a set R of reaction rules describing the interactions between the molecules, and an algorithm A that describes how the set R of rules is applied to a collection of molecules. Both sets, R and S can be defined either my an algorithm or by a mathematical expression.

The *set of molecules* S might be finite or infinite and various molecule descriptions exist:

- abstract symbols (e.g., $a, b, c \ldots$);
- character sequences (e.g., "abbagh");
- lambda-expressions (e.g., $\lambda x_1.x_2$);
- binary strings (e.g., 011011001)
- numbers (e.g., 12, 10);
- tree structures;
- combinators;
- proofs;
- etc.

The *set of reactions rules* R describes the interactions between the different molecules (i.e., who the molecules will be transformed). Reaction rules are usually written in the following form:

$$r_i = s_1 + s_2 + \cdots + s_n \longrightarrow s'_1 + s'_2 + \cdots + s'_m \tag{2.5}$$

The n molecules on the left-hand side react—when all molecules are available—and will the be replaced by the m molecules on the right-hand side. Sometimes, n is called the *order* of the reaction. As Dittrich *et al.* note, the sign "+" is not to be considered as an operator but only as a separator. In the present work, we shall rather use the "|" as a separator. In a straightforward way, further parameters and information such as energy constraints, temperature, neighboring information, etc. might be added to a reaction.

Last but not least, an algorithm , the *reactor algorithm* or the *reactor dynamics*, must be defined that determines how the set of reactions R is applied to the "soup" (also called *reactor*, *reaction vessel*, or *population*) of molecules P. P is a multiset of symbols from the set S since the molecules might be present in many examples. A reactor might also have a *spatial structure*, i.e., the position of the molecules and the reactions influences their selection. However, most often a reactor is modeled as a *well-stirred tank* (see also Figure 6.5, Section 6.2.4) where the probability that a molecule participates in a reaction is independent of its position within the reactor. A well-stirred tank is usually not spatially structured, although it is not impossible to build a spatially structured reactor with equal reaction probabilities.

Dittrich *et al.* listed the following different reaction dynamics:

- **Stochastic molecule collisions.** This is the most straightforward reaction dynamics where each molecule is explicitly simulated. The algorithm draws either a reaction and checks whether the reactor contains the right molecules to be applied or it draws a set of molecules and tries to find a reaction that might be applied. Algorithm 3 illustrates a possible implementation.

- **Continuous differential or discrete difference equations.** The dynamics of the reactor are described by means of differential equations that reflect the development of the molecule's concentrations.

- **Metadynamics.** The system of differential equations describing the dynamics is modified by adding and/or removing equations.

- **Mixed approaches.** Approaches where some molecules are explicitly simulated and others represented by their concentrations.

- **Symbolic analysis of equations.** The system of differential equations is symbolically solved (usually only possible if simple enough) and the steady-state (i.e., fixed point, limit cycle, etc.) is derived.

A comparison of the computational cost is provided in [112]. Alternatively, an artificial chemistry might also be defined as a tuple (S, I) where S is the set of particles and I the set of interactions among these particles. This definition is sometimes more natural and simpler, especially when the reactions take place in the same space as the molecules, like for example in P systems or lattice molecular systems.

Algorithm 3 Stochastic molecular collisions in a closed reactor

1: Let P be a multiset of molecules from the alphabet S.
2: Let R be a multiset of reactions.
3: **while** not finished **do**
4: Randomly choose a reaction $r = S_n \rightarrow S'_m = s_1 + \cdots + s_n \rightarrow s'_1 \cdots + s'_m$.
5: Randomly choose n molecules from P: $M_n = \{m_1, \ldots, m_n\}$.
6: **if** $M_n \equiv S_n$ **then**
7: Remove the n molecules from the reactor: P: $P = P \backslash \{m_1, \ldots, m_n\}$.
8: Add S'_m to P: $P = P \cup \{s'_1, \ldots, s'_m\}$.
9: **end if**
10: **end while**

Molecules and reactions can be defined *explicitly* or *implicitly*. An explicit molecule definition is characterized by an enumeration of the molecules such as for example $S = \{a, b, c, d\}$. On the other hand, an implicit definition provides a description on how to *construct* a molecule. A typical implicit description might be a grammar. Other examples are the set of all binary bit-strings $S = \{0, 1\}*$ or the set of prime numbers $S = \{3, 5, 7, \ldots\}$. To build systems with (usually open-ended) constructive dynamics, it is more convenient to use an implicit definition as it is usually difficult or even impossible to define all possible (or required) molecules explicitly. An explicit definition of the reaction laws is independent of the molecular structure [112]. The reactions are explicitly given, enumerated, and do not change during the experiment. An implicit definition of the reactions must refer to the structure of the molecules, which basically allows to foresee the outcome of a reaction based on the structure of the participating molecules only. Again, the implicit definition of the reactions is usually used for constructive chemistries where new components may appear that may change the system's dynamics. A system is *weakly constructive* [112] if the new components are created randomly and it is called *strongly constructive* when new components are generated through the interaction with other components. Natural chemistry is considered to be strongly constructive system [137].

Rewriting systems are systems based on a set of symbols and a set of syntactic rules used for performing replacements. The rules define under which conditions a pattern of symbols can be replaced by another (specified by the rule). Well-known rewriting systems are L-systems proposed by Lindenmeier [239] and extended by Prusinkiewicz [327]. An L-system is a

formalism that allows to efficiently describe growth, e.g., plant growth. For a particular L-system, the growth always starts from the same seed cell, called *axiom*. *Production rules* are used to describe the growth of new cells from the old cells. An illustrative example is shown in Example 2.10.1 below. A rewriting rule of an L-system can either be used directly as a reaction, or a single reaction can also consist of multiple applications of many rules. As we will see in Section 2.11, P systems use rewriting rules to model a simple kind of chemical computation.

Example 2.10.1 (L-system)
Consider the L-system as shown in Table 2.1. Starting from the axiom, the rules are then simultaneously substituted as shown in Table 2.2.

Axiom:	A
Rules:	A → aBab
	B → aA

Table 2.1: L-system axiom and rules.

Depth	String
0	A
1	aBab
2	aaAab
3	aaaBabab
4	aaaaAabab
...	...

Table 2.2: Simultaneous string substitutions.

■

In the next few sections, some of the chemistries that are more or less relevant and interesting for the present work shall be presented. For a very detailed overview on various artificial chemistries, see Dittrich *et al.* [112].

2.10.2 Some Examples

The Chemical Abstract Machine

The *Chemical Abstract Machine (CABM)* [40]—suited to model concurrent computations—is based on the metaphor used in the Γ language introduced by Benâtre and Le Métayer [37]. The Γ language is defined by a structure

of the molecules it handles and by a set of reaction rules. Solutions are represented by multisets of molecules and the reactions are multiset rewritings. The Γ language shares several features of P systems that will be presented in Section 2.11.

Most available concurrent models are based on architectural concepts such as networks of processing elements that communicate by communication channels. The problem with such a concept is that it is based on a rather rigid and geometrical vision of concurrency [40]. The chemical abstract machine is based on a radically different paradigm. The states of a machine are chemical solutions where floating molecules (i.e., agents in terms of a concurrent model) can interact according to reaction rules. The definition of the machine relies on an elaboration of the Γ language by specifying a syntax for molecules and by refining the classification of the rules. The molecules $s_1, s_2, \ldots$ are terms of an algebra. The reactor solution contains finite multisets of molecules or sub-populations of molecules, called subsolutions, that are separated by the concept of membranes (this is similar to P systems again). Berry and Boudol state:

> "The strength of the cham [chemical abstract machine] lies in the membrane notation. Membranes make it possible to build chemical abstract machines that have the power of classical process caluli or that behave as concurrent generalizations of the lambda-calulus" [40, p. 219].

The subpopulations are considered like molecules and reactions take place in parallel and independently. The reactions of the chemical abstract machine are defined by a set of transformations laws, four *general laws* [40, p. 227] (valid for every chemical abstract machine) and *specific rules* in the form of a rewriting rule:

$$m_1, m_2, \ldots, m_k \longrightarrow m'_1, m'_2 + \cdots + m'_l \tag{2.6}$$

The general laws (i.e., (1) the reaction law, (2) the chemical law, (3) the membrane law, and (4) the airlock law) describe how to apply the specific rules the a given multiset. A molecule of a population or sub-population that matches the left side of a reaction will be replaced the the right side of the reaction. Further, sub-populations develop indecently and the molecules can enter or leave a sub-population (*airlock*). To define a special chemical abstract machine, a set of specific rules will be added to the general laws.

The dynamics of the reactor nondeterministic and an arbitrary number of transformations may be performed in parallel as long as no molecule is used more than once to match the left side of a reaction rule. If more than one transformation rule can be applied to a population at a given time, a reactions is chosen randomly.

Walter Fontana's "AlChemy"

Leo Buss and Walter Fontana have extensively written about abstract chemistries, self-maintaining systems and organizations [138–140]. How can such systems be formalized? What is required to produce a self-maintaining organization? What classes of self-maintaining systems are possible? Many biological systems are self-maintaining. In a biological context, self-maintenance means that simple chemical inputs are transformed through metabolic cycles into all those molecules that an organism needs to persist. Self-maintaining system are commonly robust to

In [137], Walter Fontana, introduced the constructive (i.e., new elements are allowed to emerge) chemistry "AlChemy" that is based on the λ-calculus. He has chosen λ-calculus as a formal system because it captures these two abstractions: (1) constructivism and (2) equivalence relations.

λ-calculus[11] [30, 67, 68] is a *syntactical*—as opposed to an *algebraic*—system intended as a convenient means for writing expressions which denote functions. In λ-calculus, every expression stands for a function with a single argument. The argument of the function is in turn a function with a single argument, and the value of the function is another function with a single argument. Functions are anonymously defined by a λ-expression which expresses the function's action on its argument.

For instance, the "add-two" function $f(x) = x + 2$ would be expressed in λ-calculus as $\lambda x.x + 2$ or equivalently as $\lambda y.y + 2$ and the number $f(3)$ would be written as $(\lambda x.x + 2)3$.

In other words, a λ-expression (from [112]) is a word over an alphabet $A = \{\lambda, ., (,)\} \cup V$ where $V = \{x_1, x_2, \dots\}$ is an infinite set of available variable names. The set of λ-expressions S is defined for $x \in V, s_1 \in S, s_2 \in S$ by:

$$x \in S \quad \text{variable name} \tag{2.7}$$

$$\lambda x.s_2 \in S \quad \text{abstraction} \tag{2.8}$$

$$(s_2)s_1 \in S \quad \text{application} \tag{2.9}$$

An *abstraction* $\lambda x.s_2$ might be interpreted as a definition of a function where x is the parameter in the "body" s_2. On the other hand, the *application* $(s_2)s_1$ might be interpreted as the application of s_2 on s_1. This is commonly formalized by the so-called β-rule or reduction:

$$(\lambda x.s_2)s_1 = s_2[x \longleftarrow s_1] \tag{2.10}$$

where $s_2[x \longleftarrow s_1]$ denotes the term that is generated by replacing every *unbounded occurrence* of x in s_2 by s_1. A variable is bounded if it appears in a

[11] [140, p. 59ff] provides a small but neat "λ-calculus for tourists" introduction in the annex.

form such as $\ldots(\lambda x.\ldots x\ldots)\ldots$. Interestingly, there is no algorithm which takes as input two λ-expressions and output "YES" or "NO" depending on whether or not the two expressions are equivalent. This was historically the first problem for which the unsolvability could be proven. Of course, in order to do so, the notion of "algorithm" has to be cleanly defined; Church used a definition via recursive functions, which is now known to be equivalent to all other reasonable definitions of the notion.

In the "AlChemy" ,the molecules are represented by normalized λ-expressions whereas a reaction for two colliding molecules s_1 and s_2 is to apply s_1 to s_2:

$$s_1 + s_2 \rightarrow s_1 + s_2 + \text{normalForm}((2_1)s_2). \tag{2.11}$$

The procedure *normalForm* simply reduces its argument term to the normal form. A λ-expression which does not allow any β-reduction, i.e., has no subexpression of the form $((\lambda V.E)E')$, is called a *normal form*. Not every λ-expression is equivalent to a normal form, but if it is, then the normal form is unique up to naming of the formal arguments. In 1934, Church and Rosser proved that no expression has more than one normal form, and so any two sequences of reductions that end with a normal form give you the same normal form. This means that we can consider the normal form to be the "value" of an expression: it's what's left when we finish evaluating it, and it doesn't matter how we carry out the steps in the evaluation.

By investigating the consequences of a many-body dynamical setting of λ-expressions Fontana has generated a diversity of self-maintaining organizations with the desired properties, i.e., resistance to perturbation, extensibility, and history dependence. Fontana observed that the diversity of the population reduced quickly, often leading to only one surviving self-replicating species. The organizing structure of the surviving ensembles is called *level-0 organization*. In order to obtain more complex reaction networks, Fontana introduced additional conditions, for example, a collision was considered to be elastic if the outcome of the reaction function was equal to one of the reactants. The result of such additional conditions were reaction networks called *level-1 organizations* that were composed of a huge number of molecules. *Level-2 organization*, consisted to two or more coexisting level-1 organizations. The stable coexistence is characterized by the following two conditions: (1) the organizations are intact and (2) the interaction is moderated by intermediate species called *glue*. Note that a spontaneous emergence of level-2 organization from a randomly initialized population is extremely rare.

In his master thesis [109] (see also [110]), Pietro Sperini di Fenizio rebuilds Fontana's "AlChemy" with different, more realistic operations: the elements are effectively used up in the reactions. His model also proved to be able to support and generate self-maintaining infinite sets.

Other Chemistries

- Varela and Maturana's *Autopoietic Systems*. See [262, 434, 435].
- *Assembler Automata* such as Coreworld [335], Tierra[12], or Avida [5].
- In [398, 399], Suzuki and Tanaka present an *Artificial Cell System* (ACS) that is based on a abstract chemical system. The ACS is made up of membranes, multisets of symbols and rewriting rules. The ACS is actually a class of P systems and differs from chemical abstract machine [40] only that it adds membranes. Suzuki and Tanaka also introduce a genetic method to the artificial cell system and apply it to genetic programming.
- Tim Hutton proposed an artificial chemistry that supports self-replicating molecules [203]. These self-replicators emerge spontaneously from the random soup when the right conditions are being provided. The goal of this work was to demonstrate the evolutionary growth of complexity, although they did not achieve the goal of finding interesting design solutions (partially because the only selection pressure was towards shorter molecules).

2.10.3 Wrap-Up

Carver Mead [263] thought of systems "[...] as divided into three somewhat arbitrarily levels" that must work together. At the bottom are the *computational primitives* or *elementary functions*, then the *representation of information*, and at the top the *organizing principles*. This division corresponds to the basic definitions of an artificial chemistry introduced by Dittrich *et al.* [112]. Table 2.3 illustrates the corresponding relations.

organizing principles	$\longleftrightarrow$	dynamics
representation of information	$\longleftrightarrow$	molecules
computational primitives	$\longleftrightarrow$	reaction rules

Table 2.3: Corresponding system levels.

To the best of my knowledge, no artificial chemistry optimized for a hardware implementation has been proposed so far. Some artificial chemistries have been simulated by means of configurable hardware, however: Breyer *et al.*, for example, simulated their reaction-diffusion based ecosystem on a FPGA-based reconfigurable machine [50] called POLYP [402].

One of the main problems of artificial chemistries is definitely how to "program" them. The problem is basically identical to the problem of how

[12] `http://www.isd.atr.co.jp/~ray/tierra/`

to program massively parallel machines, i.e., how to obtain a global behavior from local interactions only. Busch and Banzhaf [56], for example, use the framework of artificial chemistries to construct an automated theorem prover (ATP). On the other hand, it is of course also possible to used evolutionary algorithms to "evolve" good chemistries.

2.11 Membrane Computing (P Systems)

Membrane computing is a new paradigm that tries to imitate the way Nature "computes" at the cellular level. It is important, however, to bear in mind that membrane computing is in no way an attempt to faithfully model biological cells. Figure 2.26 illustrates four of the main domains of natural computation (see also Chapter 2). Membrane computing takes biological cells as a source inspiration only and of course rises many interesting questions. A central dilemma is whether *in silico* or *in vivo* realizations are more promising for natural computing, but maybe this dilemma has not even to be solved as we might simple be able to combine both together.

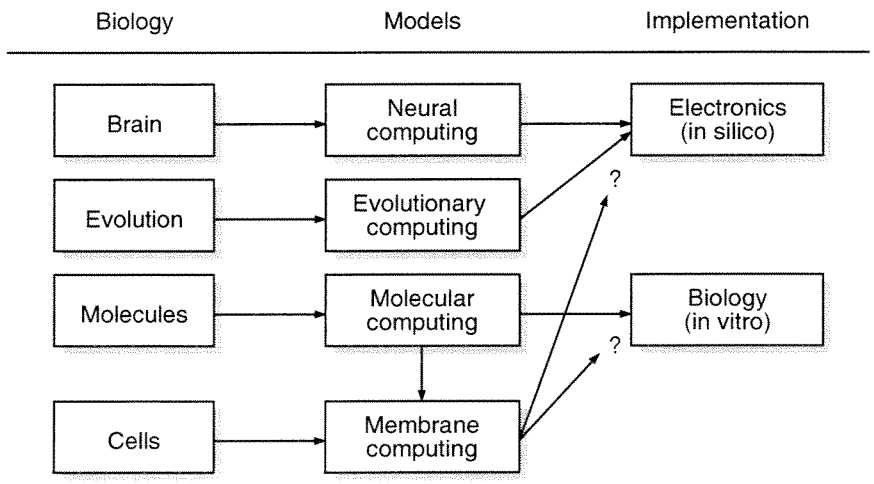

Figure 2.26: The domains of natural computing. Redrawn from [315].

Membrane computing makes use of a hierarchical membrane structure that is similar to the structure used in the chemical abstract machine as proposed by Berry and Boudol [40]. The evolution rules that transform the multisets are basically inspired by the Γ systems proposed by Banâtre [26].

In the following, an brief overview on membrane systems shall be given. A much more complete and comprehensive introduction can be found in [315].

2.11.1 Definitions

Membranes are crucial in Nature: they define compartments, they ensure that substances stay inside or outside it (*separator*), they let certain molecules pass through it (*filter*), and they form a communication structure. Every cell is enclosed in a membrane, a double layer of lipids (lipid bilayer) but is made quite complex by the presence of numerous proteins that are important to cell activity. These proteins include receptors, pores, and enzymes. The membrane forms a boundary between the cell (the "self") and its environment and is responsible for the controlled entry and exit of ions like sodium (Na) potassium (K), calcium (Ca++). The *cell membrane* is not to be confound with the *cell wall* which is the rigid structure made up of polysaccharides that provides and maintains the shape of the cell and serves as a protective barrier. The autonomous organization and self-maintenance of this boundary is an indispensable feature for living cells. This is also the basic idea behind Varela and Maturana's *autopoietic* systems [435]. Other related work comes from Gánti who proposed an abstract model of primitive cells with self-maintaining cell membranes [153, 154] and more recently from Ono and Ikegami [306, 307] who used a more biologically plausible and therefore rather different approach.

Membrane Computing (MC) or P systems, initiated by Paun [314, 316] in 1998, is a highly parallel—though theoretical—computational model afar inspired by biochemistry and by some of the basic features of biological membranes. A P system (many variations exist) consists of cell-like membranes placed inside a unique "skin" membrane (Figure 2.27).

Definition 2.11.1 (Membrane Structure)
A membrane structure *can either be represented by an Euler-Venn diagram as shown in Figure 2.27 or by a parenthesis expression:*

$$[_1\ [_2\ ]_2\ [_3\ ]_3\ [_4\ [_5\ ]_5\ [_6\ [_8\ ]_8\ [_9\ ]_9\ ]_6\ [_7\ ]_7\ ]_4\ ]_1.$$

Parenthesis expressions are, however, ambiguous representations, i.e., the same membrane structure can be represented by different parenthesis expressions. Naturally, a membrane structure can also very nicely be represented by a rooted tree as illustrated in Figure 2.28. ■

Multisets of *objects*—usually strings of symbols—and a set of *evolution rules* are then placed inside the regions delimited by the membranes. Each object can be transformed into other objects, can pass through a membrane, or can dissolve or create membranes. The evolution between system configurations is done nondeterministically by applying the rules in parallel for all objects able to evolve. A sequence of transitions is called a *computation.*

A computation *halts* when a halting configuration is reached, i.e., when no rule can be applied in any region. A computation is considered as successful if and only if it halts.

P systems are not intended to faithfully model the functioning of biological membranes, rather they form a sort of abstract *artificial chemistry (AC)*: "An artificial chemistry is a man-made system which is similar to a real chemical system" [112]. ACs are a very general formulation of abstract systems of objects which follow arbitrary rules of interaction. They basically consists of a set of molecules S, a set of rules R, and a definition of the reactor algorithm A. By abstracting from the complex molecular interactions in Nature, it becomes possible to investigate how the AC's elements change, replicate, maintain themselves, and how new components are created.

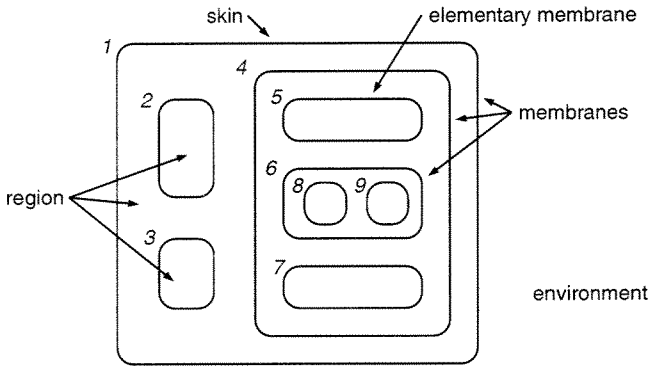

Figure 2.27: Elements of a membrane system. Source [315].

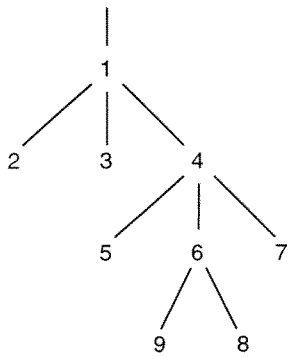

Figure 2.28: The membrane structure of Figure 2.27 represented in the form of a rooted tree.

Whereas Paun's membrane computing amalgamates in an elegant way membranes and artificial chemistries, various other systems with membranes exist. For example, Langton's self-replicating loops [232] make use of a form

of membrane (i.e., a state of the CA's cells) that encloses the program. The embryonic projects, on the other hand, uses cellular membranes to divide the empty space into a multi-cellular organism [246, 412]. Explicit membranes are not always required. The POEtic project, for example, is based on a hierarchical organization of molecules and cells with implicit separation.

In P systems (many variations exist), several cell-like membranes are placed inside a unique "skin" membrane (Figure 2.29). Multisets of *objects* — usually strings of symbols — and a set of *evolution rules* are then placed within the regions delimited by the membranes. Each object can be transformed into other objects, can pass through a membrane, or can dissolve or create membranes (only in certain P systems). The evolution between system configurations is done nondeterministically by applying the rules in parallel for all objects able to evolve. A sequence of transitions is called a *computation*. A computation *halts* when a halting configuration is reached, i.e., when no rule can be applied in any region. A computation is considered as successful if and only if it halts.

As an example, let us consider the following P system (its evolution is shown in Figure 2.30). Formally speaking, and without explaining all details, a P system is a construct

$$\Pi = (V, T, C, \mu, w_1, \ldots, w_m, (R_1, \sigma_1), \ldots, (R_m, \sigma_m)), \tag{2.12}$$

where

1. V is an alphabet. Its elements are called *objects*;
2. $T \subseteq V$ is the *output* alphabet;
3. $C \subseteq V$, $C \cap T = \emptyset$ is the set of catalysts (i.e., elements that are conserved during a reaction);
4. μ is a membrane structure consisting of m membranes; m is called the degree of Π;
5. $w_i, 1 \leq i \leq m$ are strings which represent multisets over V associated with the regions $1, 2, \ldots, m$ of μ;
6. (u, v) is an *evolution rule*, usually written in the form $u \rightarrow v$;
7. $R_i, 1 \leq i \leq m$ are finite sets of *evolution rules* over V; σ_i is a partial order relation over R_i, called *priority* relation.

An alphabet is a finite non-empty set of abstract symbols. For an alphabet V we denote by V^* the set of all strings of symbols from V (V^* is the free monoid generated by V under the operation of concatenation, mathematically speaking). The empty string is denoted by λ. The symbol δ (not in V)

is usually used to specify the dissolving action which removes the membrane where a rule containing δ was used. The objects of that region will belong to the region that was enclosing the dissolved membrane, the rules will be removed. The skin membrane is never dissolved. A membrane system that contains rules of radius greater than one, then we say that Π is a system with *cooperation*, otherwise it's a *non-cooperative system*. A particular class of cooperative systems is that of *catalytic* systems. In a catalytic system, we can either have rules of the form $a \to v$, with $a \in V, v \in (V - C)$, or rules of the form $ca \to cv$, where $c \in C, a \in (V - C)$, v a string over $(V - C)$.

The most important points to consider while evolving a P system are the following:

- All rules are applied synchronously.
- All rules are applied in a non-deterministic and maximally parallel manner. This means, that all rules that can be applied have to be applied. If several rules can be applied, the choice is made non-deterministically.
- A rule can be applied in the same step as many times as we want.

The P system of Figure 2.29 is specified as follows:

$$
\begin{aligned}
&\Pi = (V, T, C, \mu, w_1, w_2, w_3, (R_1, \sigma_1), (R_2, \sigma_2), (R_3, \sigma_3)),\\
&V = \{a, b, d, e, f\}, T = \{e\}, C = \emptyset,\\
&\mu = [_1[_2[_3\ ]_3]_2]_1,\\
&w_1 = \lambda, R_1 = \{e \to e_{out}\}, \sigma_1 = \emptyset,\\
&w_2 = \lambda, R_2 = \{b \to d, d \to de, r_1 : ff \to f, r_2 : f \to \delta\}, \sigma_2 = \{r_1 > r_2\},\\
&w_3 = af, R_3 = \{a \to ab, a \to b\delta, f \to ff\}, \sigma_3 = \emptyset.
\end{aligned}
$$

A possible evolution is illustrated in Figure 2.30.

Most membrane systems are computationally universal and several variants with an enhanced parallelism are able to solve NP-complete problems in polynomial—often even linear—time by making use of an exponential space (at least in principle). However, there are also several membrane systems which do not allow for universal computation. For example:

1. P systems with non-cooperative rules [315, p. 60–63].
2. P systems with membrane dissolution and non-cooperative rules [315, p. 70]. (The problem remains open when catalytic rules are available.)
3. Asynchronous P systems without priorities [315, p. 78].

As we will see later on, the third limitation is the most important for our case. The reason is that providing a global clock signal which synchronizes the entire membrane system is often a problem for hardware realizations. The problem is even more serious in the case of amorphous computers, which do not at all support any global signals.

Note that little is known about non-synchronized P systems, as Paun states [315]. They are actually still an active field of research and are part of the list of open problems (Problem Q3).

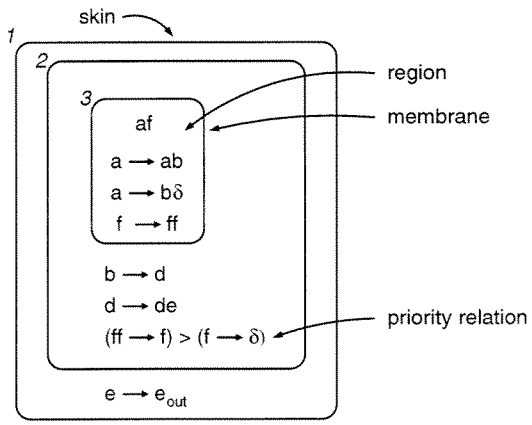

Figure 2.29: A P system generating $n^2, n \geq 1$, where n is the number of steps *before* the first application of the rule $a \rightarrow b\delta$. Source [315].

2.11.2 How to Compute with Membrane Systems

Since Turing, the concept of *computation* is well defined and its limits are well known as well (for more details see Section 2.2.1).

The question naturally arises how one can compute useful things and solve real-world problems with P systems. There exist basically two main possibilities:

1. Start with an initial configuration (i.e., rules, multisets of symbol-objects) of the membrane system. Let the system evolve until it reaches a *halting configuration*, i.e., a configuration where no further rule can be applied. The result is then associated with the entire or parts of the halting configuration. The drawback of this approach is that the system has to be set back by some means into its initial configuration before a new computation can start. Furthermore, not interaction with the environment is possible.

2. Start with an initial configuration (i.e., rules, multisets of symbol-objects) of the membrane system. Run the system continuously, pro-

vide *inputs* at precise moments in the computation, and receive the outputs related to the inputs. The advantage of this approach is that it allows to directly interact with the environment and to control, for example, a robot (equipped with sensors and motors).

Both methods are basically equivalent in the sense that an interactive computation can be "simulated" by a (possibly open-ended) sequence of halting computations.

There is a third method, or maybe rather a point of view: a membrane system can be considerer like an abstract artificial neuron (see also Section 2.4 or [308]) as a sort of detector that detects the existence of some set of conditions and that responds with a signal that communicates the extent to which those conditions have been met. The rules in the membrane system represent the conditions which transform the incoming information.

2.11.3 Wrap-Up

Although membrane systems are not faithful models of biological membranes, they provide a new and exciting territory of computer scientists to explore. The basic model is cellular, parallel, nondeterministic, synchronous, offers universal computational capabilities, dynamical hierarchies, and development. There exists an enormous variety of different membrane systems and every year new variations are being created. The interested reader is referred to the membrane systems web-site: `http://psystems.disco.unimib.it`.

The main drawbacks of the concept are the global synchronization signal assumed (which makes implementations potentially more difficult) and the fact that membrane systems have to be designed by hand, at least so far (mainly due to a lack of methodologies and tools).

Although membrane systems are mostly considered from a theoretical point of view, they offer many interesting aspects for hardware implementations, as we shall see in Section 4.8.

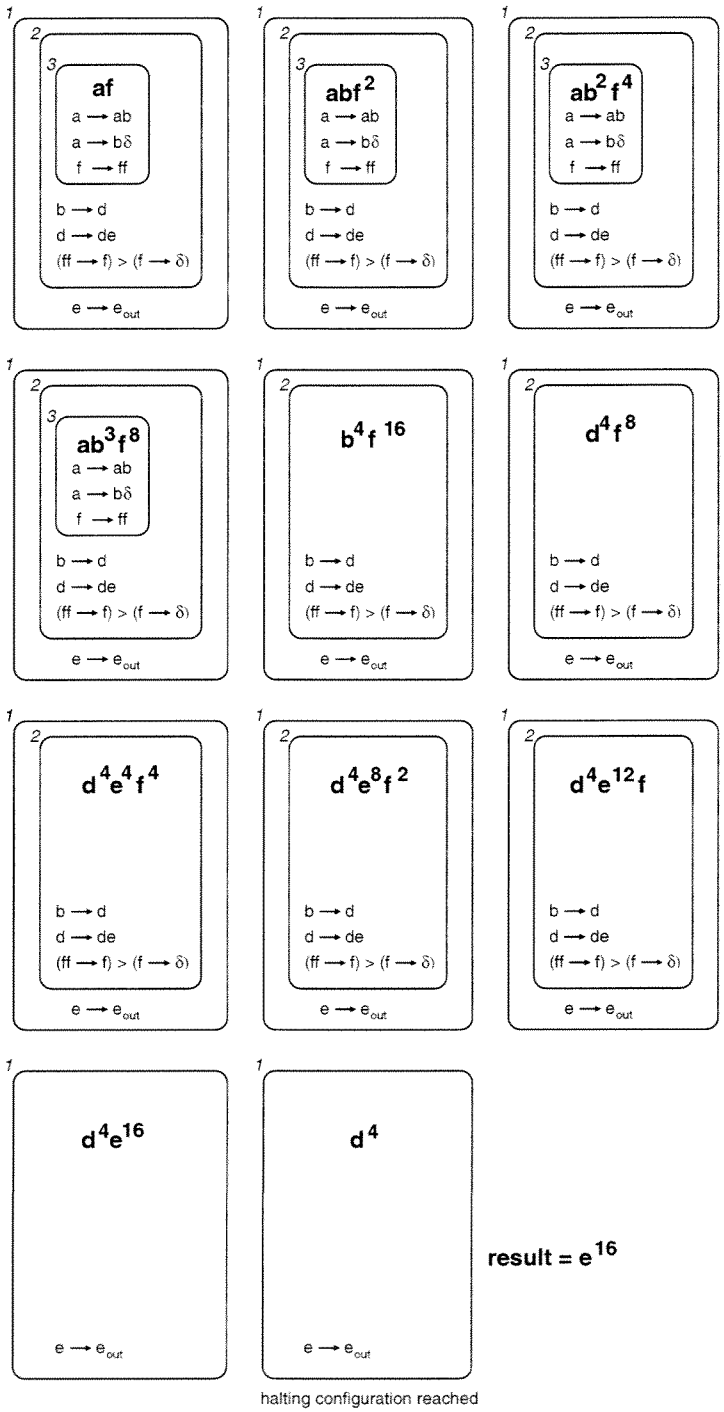

Figure 2.30: Possible evolution of the P system generating $n^2, n \geq 1$, where n is the number of steps *before* the first application of the rule $a \rightarrow b\delta$. The last step writes the results outside the skin membrane.

CHAPTER 3

From Regular to Irregular and Random Structures

> In mathematics you don't understand things. You just get used to them.
>
> *The Dancing Wu Li Masters*
> (Gary Zukav)
> John von Neumann, 1979

3.1 Introduction

It is widely believed that complex systems cannot be created by hand and that complex information processing structures can only evolve in a complex (and real) world rich in information. From that point of view, most man-made technological systems including all human institutions are probably below the threshold of true complexity, although science created highly complex systems such as airplanes, communication networks, etc. Some people make a distinction (yet not a strict one) between *complex* and *complicated* systems. Complicated systems remain understandable even though they might be built up by a large number of components. Complex systems, on the other hand, are not understandable and the closer one takes a look at them, the more details become visible. Steve Grand writes: "Something is complex if it contains a great deal of information that has high utility, while something that contains a lot of useless or meaningless information is simply complicated" [164, p. 118]. His view seems to be rather oversimplified and is not a very accurate one. Note that the distinction between complex and

complicated systems also heavily depends on the viewpoint and on the current knowledge that is available on the system. Quite naturally, one might ask whether mankind will one day be able to bootstrap, i.e., initiate, the automatic construction of complex systems. Engineering has certainly created many extremely complicated and impressive systems, but they are all "understandable". Large integrated circuits built up from millions of transistors are a good example for a complicated system whereas a biological cell might be considered as a complex system. Thus, the ultimate goal for the future is the automatic creation without an external supervisor of structurally complex (and not just complicated) artefacts by means of new approaches.

John von Neumann is not only one of the spiritual fathers of modern computing science but also the first who conceived a *cellular automaton (CA)* His seminal work on self-reproducing automata was completed and described in details by A. Burks [445]. In the 21^{st} century, Stephen Wolfram's "A New Kind of Science" [465] brought cellular automata in the reach of *John Q public*. Cellular automata are discrete dynamical systems whose behavior is completely specified in terms of a local relation. Usually, the cellular space can be thought of as a uniform grid (in one or more dimensions) of cells. For most elementary automata, the cell's state is binary and determined by a rule (i.e., a boolean function) whose inputs are the immediate neighboring cell's states as well a its own state. Usually, the cell's states are updated synchronously. When each cell is governed by the same rule, a CA is called *uniform*, otherwise *nonuniform*. The main advantage of cellular automata is that its components are extremely simple, locally interconnected, and that the ensemble relies on a massive parallelism. All this comes with a serious drawback: the nontrivial problem of programming parallel systems. There is definitely no generally applicable method to determine which rules to use for which problem, reason why people often fall back on using evolutionary algorithms [371].

Noise and randomness — which are rarely present in cellular automata (see [378] for a counter-example) — are an absolutely crucial and ubiquitous element for many Natural system. In engineering, they have for a long time be considered as an awkward and disturbing issue, but it has become more and more clear in recent years that noise and randomness very often plays an absolutely crucial role in nonlinear and complex dynamical systems, coupled oscillators, pattern formation, etc. Noise might for example initiate symmetry breaking and generally plays its cards in instable system states and bifurcations. And even more generally, noise and chaos can help order arise from disorder [55].

The interested reader is referred to [3, 23], for example. For stochastic resonance, chaos and the ubiquity of noise see for example [177,210,461,462, 475].

The main goal of this chapter is to present several types of regular and random networks with different node updating schemes. In order to perform various tests, a random boolean network MATLAB toolbox has also been implemented.

3.2 Organization and Hierarchies

The section will feature some general considerations about the hierarchical organization of systems. It should mainly be considered as an introduction and motivation of why I have used hierarchical systems, namely membrane systems.

3.2.1 Multiple Dynamical Hierarchies

Hierarchical composition is ubiquitous in any physical and biological system, is however not limited to living systems only. One example of a hierarchical system is a social insect system such as the system ants. The ants are the agents at the lowest level and they act in their environment. At the higher level one can find the ant colony with its nest. Another good example is the hierarchical organization of a biological organism that is constructed from atoms, molecules, organelles, cells, and organs (to be continued with organisms, populations, ecosystems, biosphere). Or even a computer with its software is organized hierarchically (transistors, gates, logic blocks, ..., CPU, ..., assembler language, high-level language, application). And modularity, for example, seems critical related to evolution through natural selection [238].

In Nature, the nonlinear dynamics at each level of description generates *emergent structure*, and the nonlinear interactions among these structures provide a basis for the dynamics at the next higher level [361]. Of course, a central and philosophical question with the ontological nature of emergent levels is whether they have been created for academic purposes only or whether they qualitatively represent reality? To questions can sometimes be answered by observing whether the upper levels can be derived from the lower levels, which inescapable also brings us to the problem of *reductionism*, which shall not be considered here.

But hierarchy can mean many things and there does not exist a generally accepted formalism, although several researchers tried to establish definitions (see for example [167] or [302] for a rather different approach). Others noted that (see for example [20, 257]) the generation of dynamical hierarchies from a formal system is not trivial. Especially, higher order structures (more than two) are difficult to achieve. Mayer and Rasmussen [257] as well as Fontana [138–140] make use of chemical systems to create hierarchical structures.

Hierarchical modularity is a familiar characteristic of a large class of natural dynamical systems. A rather intuitive interpretation of modularity is that the interactions between the subsystems are sparse compared to interactions within subsystems. This should however not lead to the conclusion that the interactions between the modules are less important or even unimportant.

As Watson notes, modular systems that are decomposable but not separable — also called systems with *modular interdependancy* — can form hierarchical systems where all levels or organization are significant. A known problem is that systems that have rather strong dependencies between the modules can be difficult to evolve. Especially, classical hill-climbing is unable to resolve inter-module dependencies, but compositional mechanisms, such as sexual recombination between diverse lineages in a subdivided population are better able to resolve intermodule dependencies [452].

The problem of (self-) assembling multi-levelled hierarchical structures is an open one [34]. In [333] proposed that it may be impossible to extend the levels in a hierarchy without adding to the complexity of the base unit. This statement naturally raised a number of potentially interesting questions (see also [168,332]):

- How much complexity do the base units require?
- How is this complexity related to the number of hierarchical levels that can be constructed?
- Is there a minimal complexity necessary (threshold)?
- Is the physical world limited in the number of possible hierarchical levels?

Dorin and McCormack [114] propose a rather trivial (in the sense that the hierarchy is not interesting) model of infinitely-levelled, self-assembling dynamical hierarchy that arises from the interaction of geometric primary elements (equilateral triangles) with a fixed complexity. The triangles are laid out on a planar triangular grid and the entire system is updated synchronously in discrete time steps. The update rules are rather simple: at each time step, each triangle many be shifted randomly to unoccupied neighboring location (if one is available). After the movement, all triangles are examined to see if they neighbor any others. If they do, the two triangles will bond to one another with a fixed probability b. If two neighboring triangles are already bond, they will dissociate with probability d. Dorin and McCormack show that hierarchical structures (although useless) are self-assembled. The model they propose meets all the criteria to disprove Rasmussen *et al.Ansatz* [333] (see also [168, 332]). They further propose that a more formal approach — inspired by the information content of the

hierarchical structures — would required to deal with the emergence of new properties and structures.

Sometimes, a distinction is made between *structural* and *dynamical hierarchies* (see for example [238]). The difference lies in the mechanisms which produce the hierarchy: *structural hierarchy* is usually imposed and engineered through the separation of different parts into different modules (e.g., designing software), whereas *dynamical hierarchies* are the result of different lower-level components (e.g., proteins that build higher order entities).

Baas [20] introduces a general framework for the study of emergence, hierarchies, and hyperstructures. This framework tries to unify the different definitions of, and approaches to the study of emergence and hierarchies using the notion of hyperstructures. The creation of such a structure is illustrated in Figure 3.1. This new notion seems to provide us with a tool to describe a complex system in a reasonable way.

However, despite all the efforts made with the self-organization and creation of hierarchies, the following questions remain open:

- How can hierarchies emerge through the interaction of (simple) components?
- How can this process be formally defined?

3.2.2 An Example: The Subsumption Architecture

The classical approach to building control systems for robots was to decompose the problem into several functional modules. This approach, however, usually quickly runs into scalability and extensibility problems, is not robust to failures, and is not easily realizable for the integration of multiple sensors and for achieving many competing goals.

Rodney Brooks proposed another approach [52]: he decomposes the problem of building and controlling autonomous robots into various layers of *levels of competence* that possess pre-wired behaviors. The architecture is called the the *subsumption architecture*. Each higher level basically contains a subset of each lower level of competence. The overall systems behavior is the result of the interaction of the simple behaviors that operate asynchronously. The layers of the architecture are basically composed of networks of finite state machines augmented with timer that allow to change states after a programmed period of time. Each state machine can be considered as a separate agent that has input and output lines and that can send and receive messages. The agents also accept a *suppression* and a *inhibition* signal. The suppression signal can override a normal signal whereas the inhibition causes the output to be completely inhibited.

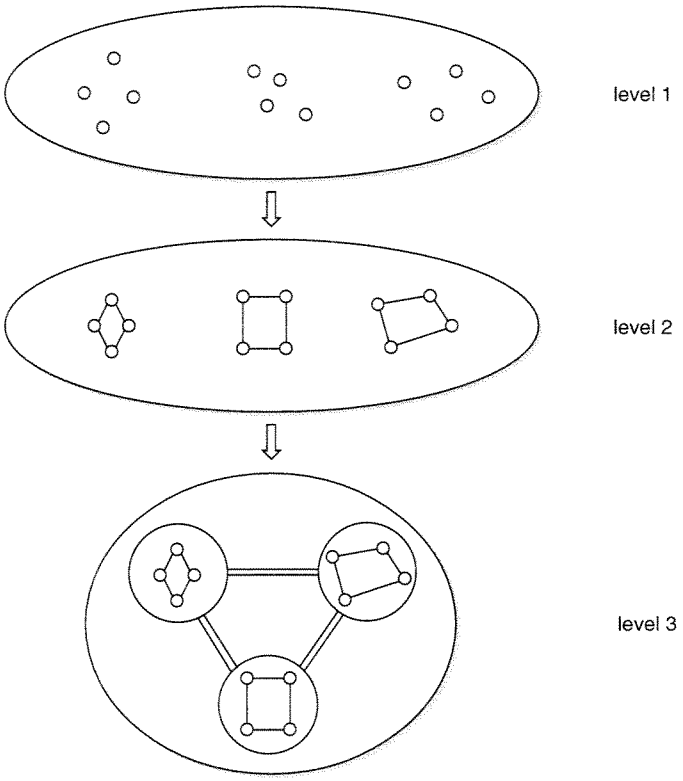

Figure 3.1: Creation of a three-level hyperstructure.

It is also important to note that the finite state machines are the only processing units in the architecture and that there are therefore no symbols used. The architecture has been further developed later by Brooks and others. Several robust subsumption architectures have been proved to work on real-robots and in real-time. However, adding some new behaviors or modifying the architectures is all but easy and is usually done by hand.

The subsumption architecture is especially interesting from the point of view of its hierarchical organization.

3.3 Biological and Artificial Cells

"The simplest living cell is so complex that supercomputer models may never simulate its behavior perfectly. But even imperfect models could shake the foundations of biology." Thus the beginning of a recent article in the Scientific American [160]. The article emphasizes that most attempts to create artificial life or to faithfully model biological systems suffered from a tremendous number of degrees of freedom. The system's parameters could then be tweaked to produce almost any desired behavior. Models are of-

ten so complicated that they have absolutely no ability to predict anything. However, often the goal is not to obtain a perfect prediction but rather a reliable approximation. One of the early findings—intuitively expected—of sophisticated cell simulations is the principle of robustness. "To survive and prosper, cells must have backup systems and biological networks that tolerate interference." Simulations at the University of California at San Diego done by Bernhard O. Palsson's team successfully predicted for example that *E. coli* is optimized for growth, not energy production.

On the other hand, Masaru Tomita [419] believes that the study of the cell will never be complete unless its dynamic behavior is understood. He suggests that the complex behavior can only be understand by means of computer models that allow to undertake complete simulations. The construction of realistic living cells *in silico*, thought intractable so far, has become realistic. "[W]e are just opening the door to this new area of biological research in the 21st century."

In a recent Nature article, Drew Endy and Roger Brent [124] also emphasize that "[p]ast efforts to model behaviour of molecular and cellular systems over absolute time typically were qualitatively incomplete or oversimplified compared to available knowledge, and qualitatively incomplete in the sense that key numbers were unknown." It is estimated that complete *E. coli* simulations could run, perhaps, on a single processor system by 2020. One of their central messages is that more information is needed:

1. Before we can model cellular systems quantitatively, we first need to overcome gaps in understanding.

2. We need to understand better the physics of some intercellular phenomena.

3. Even for biological systems in which all the components are known, we seldom understand precisely how they interact to make the process work.

4. We need to learn to make better use of physics to constrain models, and to define key experiments.

5. We need to devise new experimental methods for obtaining quantitative data about biological processes.

Whatever model is chosen for simulation, it always embodies a physical and logical level of resolution. For a better understanding of the model, it would be necessary to allow for transition among different levels of resolution.

Several attempts tried to model and specify biological systems by means of logical systems and formal languages. Duan *et al.*, for example, propose

a *hybrid projection temporal logic (HPTL)* that allows to model, analyze, and verify biological systems. Thereby, biological systems are considered as a combination of both discrete events and continuous dynamical processes, so called *hybrid systems*. The basic systems model used is a hybrid version of the X-machine, originally invented by Eilenberg in 1974 [122] and further developed by various people[1].

Other formal models come from Bell [36] and Duan *et al.* [116], who propose a specification language for analyzing and verifying biological systems. Sipper proposed the *cellular computing* approach [372], which is based on the following three principles:

- Simplicity: Simple processors—referred to as cells—comprise the basic units of computation.
- Vast Parallelism: There are a vast number of cells operating in parallel.
- Locality: The connections between cells are local.

Movable Finite Automata (MFA) [162,415] models are similar to CAs but the key feature allowing for greater biophysical realism is that the automata are allowed to move about and to interact with one another.

3.4 Cellular Automata as a Showcase of Cellular Systems

Cellular automata (CA) were originally conceived by Ulam and von Neumann in the 1940s to provide a formal framework for investigating the behavior of complex, extended systems [445]. CAs are dynamical systems in which space and time are discrete. A cellular automaton consists of a regular grid of cells, each of which can be in one of a finite number of k possible states, updated synchronously in discrete time steps according to a local, identical interaction rule. The state of a cell is determined by the previous states of a surrounding neighborhood of cells [416, 463].

In cellular automata, objects that may be interpreted as passive data and objects that may be interpreted as computing devices are both assembled out of the same kind of structural elements, and subject to the same laws; computation and construction are just two possible modes of activity.

From point of view of physics, a CA is fully discrete classical field theory where space and time are discrete and where each discrete lattice point has only a finite number of possible discrete values. Probably the most important property of CA's is that they emulate the *spatial locality* of physical law: the state of a given lattice only depends on the previous state of the neighboring

[1] See `http://www.dcs.shef.ac.uk/~mps/xmachines/` for more details.

sites. Discrete lattice models have been used in statistical mechanics since the 1920's. One of the best-known models is certainly the *Ising model* [207]—one of the pillars of statistical mechanics, where each site in a lattice contains a *classical spin*, i.e., a particle that can be in one of two classical states. Neighboring sites have an energetic preference to be the same value. The Ising model has been invented to describe phase transition in magnetics, but it also describes gas-liquid phase transition. More recently, the Ising model has been used to model phase separation in binary alloys and spin glasses. In biology, it can model neural networks, flocking birds, or beating heart cells. It can also be applied in sociology.

The main problem with cellular automata – or more generally with all massive parallel systems – is to find the rules of each cell (i.e., the program of each node in the case of massive parallel machines) that allows to obtain a global behavior from local interactions only. Most cellular automata are either programmed by hand – which might really be very tedious and challenging – or they are evolved by means of evolutionary algorithms (see for example [45, 60, 371]).

3.5 The 2D-Firefly Synchronization Task

The mutual synchronization of oscillators—a both surprising and interesting phenomenon—in living things and nature is ubiquitous (see for example [161]): dancing (natural response to music), pacemaker cells in the heart, the circadian clock of multicellular organisms (consists of multiple autonomous single-cell oscillators) [429], the nervous system that controls rhythmic behavior such as breathing, running and chewing, synchronous flashing of fireflies, choruses of cicadas or crickets [450], etc.

An oscillator can be defined in a loosely manner as a system that executes a periodic behavior. The pendulum is probably the best example of a periodic behavior in space and time. Electronic circuits offer many different types of oscillators that produce an output signal of a specific frequency. Often, a very stable mechanical oscillator, such as a specially prepared quartz crystal, may be coupled to an electronic oscillator to enhance its frequency stability.

Once the behavior of a single oscillator is understood, the more complex behavior of coupled oscillators can be investigated, although the equations governing their behavior usually become quickly intractable. Strogatz and Stewart [394] write that the most familiar mode of organization for coupled oscillators is synchrony. "One of the most spectacular examples of this kind of coupling can be seen along the tidal rivers of Malaysia, Thailand and New Guinea, where thousands of male fireflies gather together in trees at night and flash on and off in unison in an attempt to attract the females

that cruise overhead. When the males arrive at dusk, their flickerings are uncoordinated. As the night deepens, pockets of synchrony begin to emerge and grow. Eventually whole trees pulsate in a silent, hypnotic concert that continues for hours" [394]. Synchronization in living things usually emerges spontaneously and through cooperative behavior: if a few individuals happen to synchronize, they tend to exert stronger influence on their common neighbors. See Buck and Buck [54] for a review of various theories and information about synchronous fireflies.

Another well known example is the synchronization of menstrual cycles among women friends or roommates [260]. There exist various ideas about the mechanism of synchronization, but an experiment in 1980 has shown that it might have something to do with sweat [349].

In this section, we concentrate on the two-dimensional firefly synchronization task for non-uniform CAs (each automaton may have a different rule) and random boolean networks. The two-dimensional version of the task is in principle similar to the one-dimensional version, although the speed of synchronization is in general much faster.

3.5.1 The Synchronization Task for Synchronous Cellular Automata

The one-dimensional synchronization task for synchronous CA was introduced by Das *et al.* [91] and studied among others by Hordijk [201] and Sipper [371]. In this task the two-state one-dimensional CA, given any initial configuration, must reach a final configuration, within M time steps, that oscillates between all 0s and all 1s on successive time steps. The whole automaton is then globally synchronized. Synchronization comprises a non-trivial computation for a small-radius CA: all cells must coordinate with all the other cells while having only a very local view of its neighbors. The existence of the global clock, though not without consequences as we will see later, should not lure us into thinking that the task is straightforward. Obviously, the fact that the whole computation should only occur within a two-state machine prevents any counting.

In this section, we concentrate on the the two-dimensional version of the task for non-uniform CAs and random boolean networks. The two-dimensional version of the task, see Figure 3.2, is identical to the one-dimensional version except that the necessary number of time steps granted to synchronize is not any more in the order of N, the number of cells in the automaton, but in the order of $n + m$ where n, m are the size of each side of the CA. Thereby, the speed of synchronization is much faster. Non-uniform CAs are CAs where each automaton may have a different rule.

Obviously, in the non-uniform case there is an immediate solution consisting of a unique 'master' rule, alternating between '0' and '1', whatever

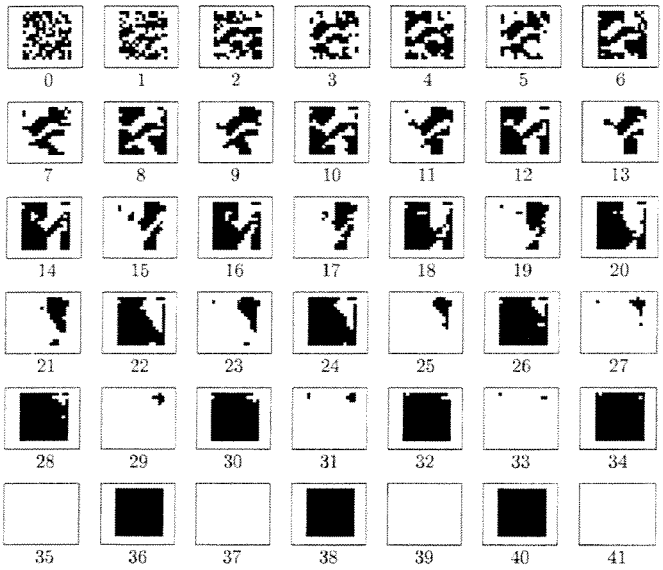

Figure 3.2: The two-dimensional synchronization task for synchronous CAs. Source: [371].

the neighborhood, and all other cells being its 'slave' and alternating according to its right or up neighbor state only. However, Sipper [371] used non-uniform CAs to find perfect synchronizing CAs only by means of evolution. It appeared that this "basic" solution was never found by evolution and, in fact, the "master" or "blind" rule 10101010 (rule 170) in one dimension, was never part of the evolved solutions. This is simply due to the fact that this rule has to be unique for the solution to be perfect, which is contradictory to the natural tendency of the evolutionary algorithm used, as we demonstrated in [62]. In this paper, we only use evolved solutions to this task.

3.5.2 Asynchronous Cellular Automata

In section 3.5.1 the CA model is synchronous as is traditionally the case. But this feature is far from being realistic from a biological point of view as well as from a computational point of view. As for the former, it is quite evident that there is no accurate global synchronization in nature. As for the latter, it turns out to be impossible to maintain a large network of automata globally synchronous in practice. Besides, it is interesting to try to delineate what part of the computation relies on the synchrony constraint.

Ingerson and Buwel [206] compared the different dynamical behavior of synchronous and asynchronous CAs; they argued that some of the apparent self-organization of CAs is an artifact of the synchronization of the clocks.

Wolfram [464] noted that asynchronous updating makes it more difficult for information to propagate through the CA and that, furthermore, such CAs may be harder to analyze. Sipper *et al.* [379] and Tomassini and Venzi [418] evolved robust asynchrnous CAs for the density task. Asynchronous CAs have also been discussed from a problem-solving and/or artificial life standpoint in references [41, 181, 211, 301, 358, 387].

All these works devoted to asynchronous cellular automata only concentrated on the study of the effects but not on correcting asynchrony or dealing with it. From a theoretical computer science perspective, Zielonka [476, 477] introduced the concept of asynchronous cellular automata. Though the question attracted quite some interest [88, 228, 326], the essential idea behind them was to prove that they were "equivalent" to synchronous CA in the sense that, for instance, they recognize the same trace languages or could decide emptiness. From these two fields, we thus knew that asynchronous CA are potentially as powerful as synchronous CA, computationally speaking[2], and, nevertheless, that most of the effects observed in the synchronous case are artifacts of the global clock.

In [60, 61] Capcarrère proposed asynchronous CAs exhibiting the same behavior as synchronous CAs through both design and evolution. The main idea behind these models was that time is part of the information contained in a CA configuration. Hence, suppressing the global clock constraint is equivalent to suppressing information! Thereby, and quite naturally, if we are to maintain the capability of the CA, then we must compensate for that loss of information by adding extra states. As shown then, it is possible to design a CA with $3 * q^2$ states working asynchronously which simulates perfectly a given q-state CA. This is of course a highly exciting result, which seems to have found independently by Nehaniv as well [288]. Tolerating some loss of information, we have shown that it was possible to evolve self-correcting 4-state CA to do the synchronization task under low asynchrony.

As Gacs [150, 151] reminds us, asynchrony may be considered as a special case of fault-tolerance. However, even though this consideration is nice in its generalization (i.e., a fault-tolerant CA is also asynchronous), it eschews a lot of potential optimization.

3.5.3 Implementations and Experiments

To better compare the different simulations and implementations, we define a number of metrics (some of them adapted from [62]).

Let's first define N to be the total number of cells. We have $N = n * m$ where m, n are the length of the sides of the two-dimensional CA considered. We can now define the frequency of transitions, ν, as the number of borders between homogeneous blocks of cells having the same genotype, divided by

[2] In the traditional sense, not in the sense of visual computation.

the number of distinct couples of adjacent cells. Adjency is here taken along the neighborhood used, that is the von Neumann neighborhood. Thus ν is the probability that two adjacent cells have a different rule (Equation 3.1).

$$\nu = \frac{1}{2(n*m)} \sum_{i=1}^{n} \sum_{j=1}^{m} \left[R_i, j \neq R_{(i+1 \mod n, j)} + R_i, j \neq R_{(i, j+1 \mod m)} \right] \tag{3.1}$$

can also define a more global measure of the diversity of the different genotypes encountered, an entropy, H (Equation 3.2).

$$H = \sum_{r \in \Gamma} q(r) * log \left(\frac{1}{q(r)} \right) \tag{3.2}$$

where Γ is the set of all possible genotypes (rules), and $q(r)$ is the global proportion of the genotype (rule) r in the cellular automata. Obviously, H takes on values in the interval $[0, \log n]$ and attains its maximum, $H(x) = \log n$, when x comprises n different genotypes. H is usually normalized to take values in the interval $[0, 1]$.

Fireflies synchronize according to the "pulse coupled" system [274], i.e., they interact only when one sees the sudden flash of another. The firefly then shifts its rhythm accordingly. "Pulse coupling is difficult to handle mathematically because it introduces discontinuous behavior into the otherwise continuous model and so stymies most of the standard mathematical techniques" [394]. Mathematicians have turned to the theory of symmetry breaking to tackle the complex problems that arise when identical oscillators are coupled. Computer scientists, on the other hand, often use cellular automata, irregular networks and grid arrays or more complex agent-based models to analyze the behavior of interacting elements.

To verify our approach, we recently implemented a two-dimensional CA on the BioWall that is able to solve the synchronization task. For further technical information regarding this implementation, the reader is referred to Section 4.3.4.

The model used consists of a two-state, non-uniform CA, in which each cell may contain a different rule. The cells rule tables are encoded as a bit-string, known as the genome, that has a length of $2^5 = 32$ bits for our 2D CA since the binary CA has a neighborhood of 5 (see also Figure 3.3).

Rather than to employ a population of evolving CAs, our algorithm (see Algorithm 4) evolves a single, non-uniform CA of a given size, whose rules are initialized at random. Initial configurations are then randomly generated and for each configuration the CA is run for M time steps, where n, m are the size of each side of the CA. Each cell's fitness is accumulated over C initial configurations. The (local) fitness score for the synchronization task is assigned to each cell by considering the last four time steps ($M + 1$ to

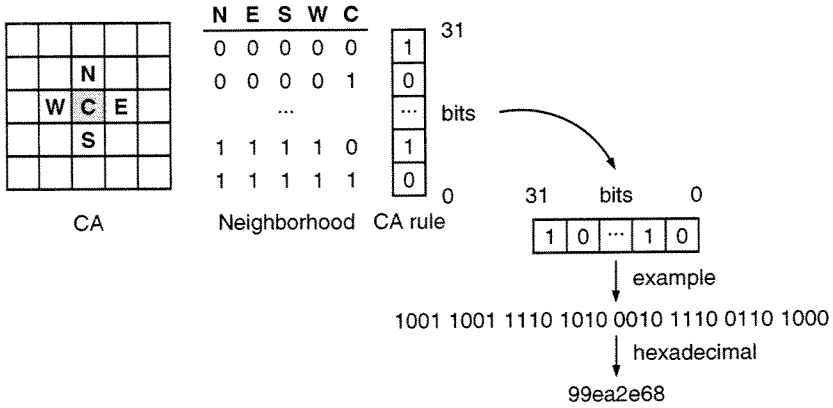

Figure 3.3: Von Neumann neighborhood of our two-dimensional CA and cell rule encoding.

$M + 4$): if the sequence of states over these steps is precisely $0 - 1 - 0 - 1$, the cell's fitness score is 1, otherwise this score is 0.

After every C configurations the rules are evolved through crossover and mutation. This evolutionary process is performed in a completely local manner, that is, genetic operators are applied only between directly connected cells. Unlike standard genetic algorithms, where a population of independent problem solutions globally evolves, our approach involves a grid of rules that co-evolves locally. The CA used performs computations in a completely local manner, each cell having access to its immediate neighbors' states only. In addition, the evolutionary process is also completely local, since the application of the genetic operators as well as the fitness assignment occurs locally. Using the above-described cellular programming approach (see also Algorithm 4), we have shown that a non-uniform CA of radius 1 can be evolved to successfully solve the synchronization task.

As an example, Figures 3.5 and 3.6 shows the evolution of the fitness as well as the number of rules in function of the number of evolutionary phases for a small 4×4 cell cellular automata (toroidal). A solution that perfectly synchronizes the 16 cells from any initial configuration (there are $2^{16} = 65536$) has been found with the following five rules: `99ea2e68`, `99e82e68`, `81f2a5fa`, `fe7aeb98`, `bf82fa06` (see Figure 3.3 for the rule encoding). The rules are arranged in the cellular automata as illustrated in Figure 3.4.

Note that we exhaustively tested all 65536 configurations for this very small cellular automaton. As stated in Section 3.5.1, there is a trivial solution with two rules (one cell being the "master" and all the other cells being 'slaves'), but it is very unlikely that this configuration is found by evolution since this rule has to be unique for the solution to be perfect. The above example has a ν of 0.563, the entropy H is 0.562.

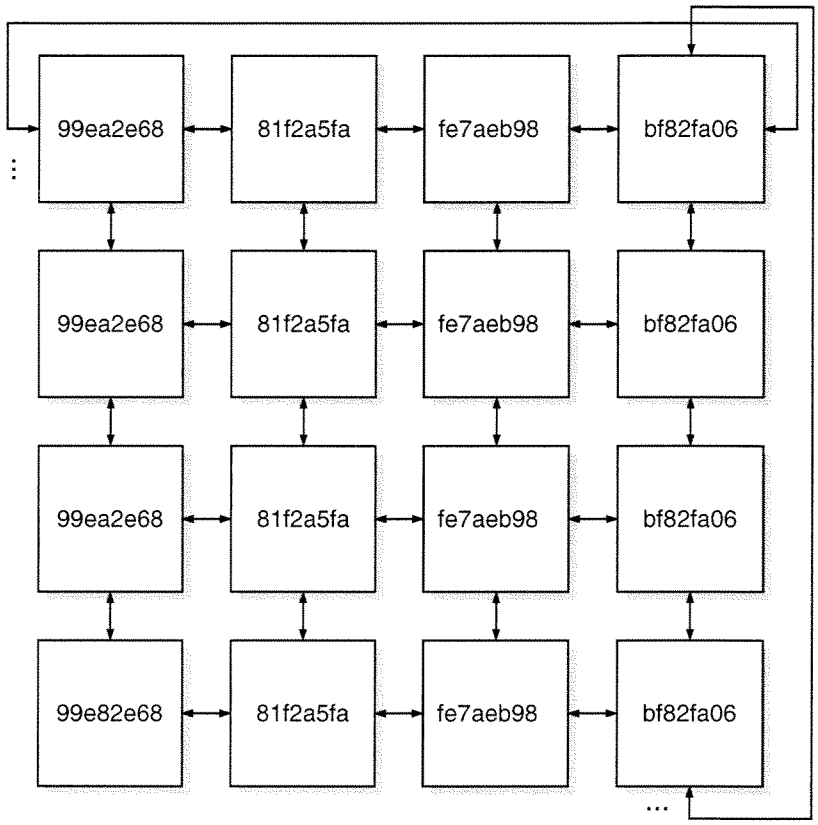

Figure 3.4: The five rules which allow to perfectly synchronize a 4×4 cell cellular automata (toroidal).

3.5.4 Asynchronous Co-Evolution

In the previous section we described the co-evolution of a synchronous CA where the genetic algorithm in each cell is synchronized by means of a global signal: all cells evaluated the fitness at the same time, replaced the rules at the same time, etc. It would obviously be interesting to use some kind of "asynchronous" co-evolution where cells still operate synchronously, but where the genetic algorithm operates asynchronously and randomly in time. This would allow to remove the GA's global synchronization signal.

We used the method as shown in Algorithm 5. For related work see also [159, 418].

Our simulations (with $T = 10N$, $D = N$) have revealed that the CA is still able to synchronize perfectly, although more different rules are required and synchronization seems slower in general. Evolution found eight different rules for the example of the previous section. Naturally, the probability that two adjacent cells have a different rule was higher: $\nu = 0.969$. The entropy

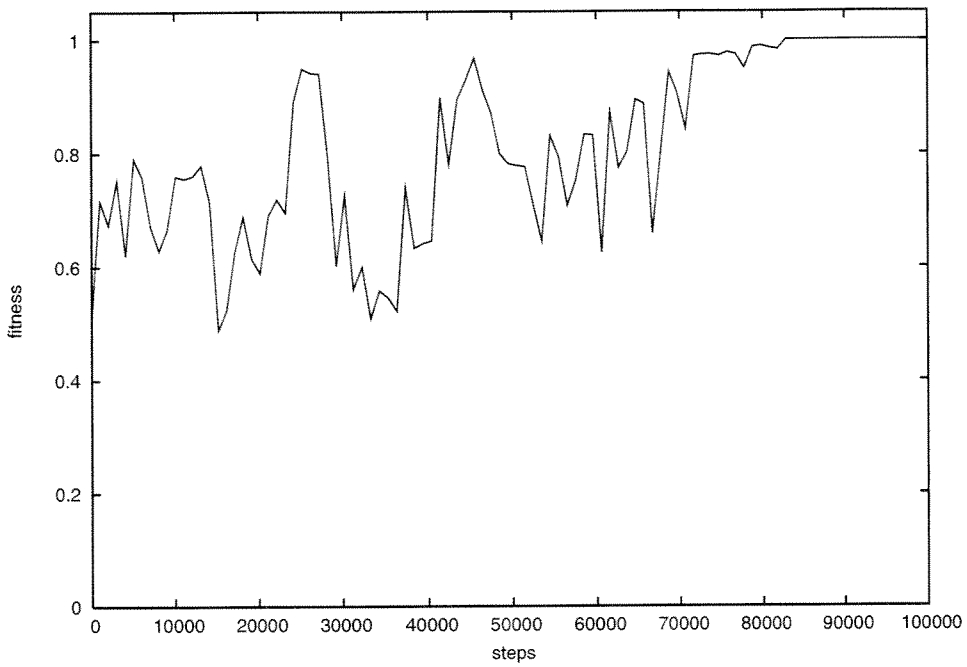

Figure 3.5: The CA's fitness of a small 4×4 cell CA (means over several runs). The CA successfully synchronizes for any initial configuration using five different rules.

H was 0.938.

3.5.5 The Firefly Java Program

In order to test various rules and configurations for the firefly synchronization task, we[3] implemented a Java program that allows to easily simulate firefly systems. The program is freely available here: `http://www.teuscher.ch/christof/firefly.html`. An illustration of the program is given in Figure 3.7. It is possible to find a solution to the synchronization task by means of the co-evolving genetic algorithm as presented above (Algorithm 4) but one can also perform an exhaustive search ("brute force find").

3.5.6 Wrap-Up

In this section we considered the issue of evolving two-dimensional cellular automata to solve the firefly synchronization task. The task has been solved using a co-evolutionary genetic algorithm that performs computations in a

[3] I am indebted to Pierre-André Mudry for the implementation.

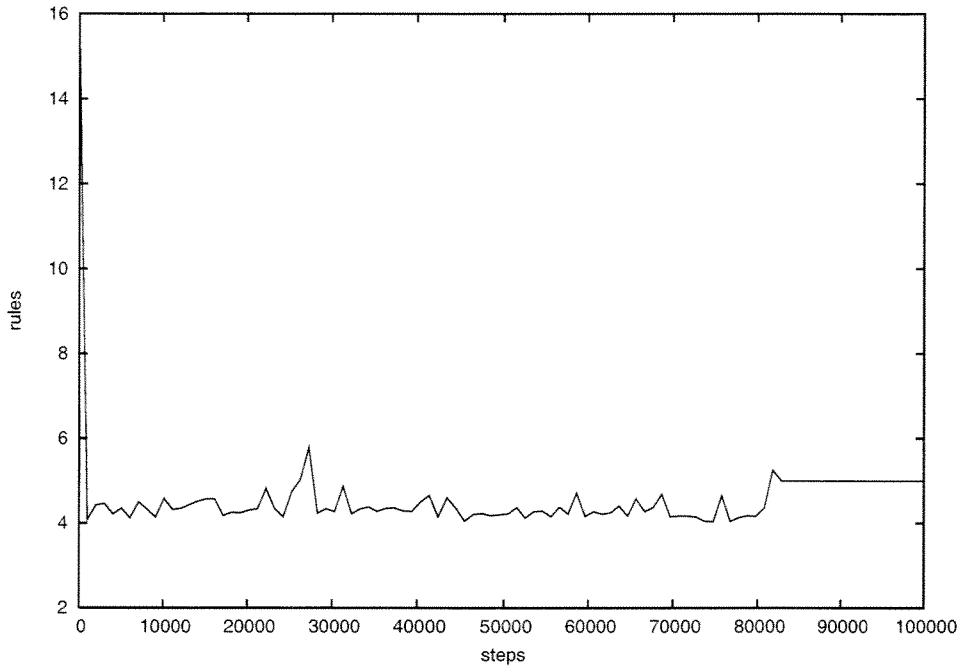

Figure 3.6: The number of different rules of a small 4×4 cell CA (means over several runs). The CA successfully synchronizes for any initial configuration using five different rules.

completely local manner, each cell having access to its immediate neighbors' states only.

Our experiments have shown that the cellular programming based approach is very efficient, easily finds a solution for classical two-dimensional cellular automata, and is easily implementable on cellular hardware (see Section 4.3.4) since there are—except the global clock signal—no other global signals needed.

In Section 3.7 we will see, that the firefly synchronization also works for random boolean networks.

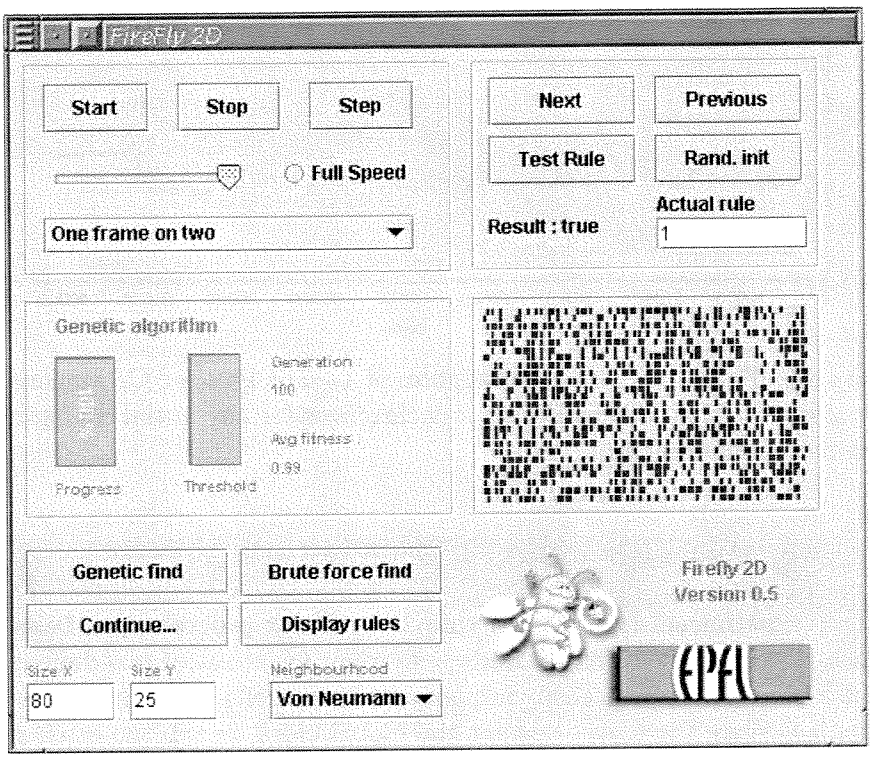

Figure 3.7: The firefly Java program allows to find a solution to the synchronization task by means a co-evolving genetic algorithm. An exhaustive ("brute force") search can also be performed.

Algorithm 4 Cellular programming pseudo-code for the synchronization of a two-dimensional cellular automaton

```
1: for each cell i in CA [in parallel] do
2:    Initialize rule table of cell i randomly
3:    f_i = 0 [fitness value]
4: end for
5: c = 0 [initial configurations counter]
6: while not done do
7:    Generate a random initial configuration
8:    Run CA on initial configuration for M = 2(m + n) time steps
9:    for each cell i in CA [in parallel] do
10:      if cell i is in the correct final state then
11:         f_i = f_i + 1
12:      end if
13:   end for
14:   c = c + 1
15:   if c mod C = 0 [evolve every C configurations] then
16:      for each cell i [in parallel] do
17:         Compute nf_i(c) [number of fitter neighbors]
18:         if nf_i(c) = 0 then
19:            Rule i is left unchanged
20:         else if nf_i = 1 then
21:            Replace rule i with the fitter neighboring rule, followed by mu-
               tation
22:         else if nf_i > 1 then
23:            Replace rule i with the crossover of the two fittest neighboring
               rules (randomly chosen, if equal fitness), followed by mutation
24:         end if
25:         f_i = 0
26:      end for
27:   end if
28: end while
```

Algorithm 5 Cellular programming pseudo-code for the synchronization of a two-dimensional cellular automaton using asynchronous co-evolution

```
1: for each cell i in CA [in parallel] do
2:    Initialize rule table of cell i randomly
3: end for
4: while not done do
5:    Generate a random initial configuration
6:    for D steps do
7:       Run CA on initial configuration for T time steps
8:       Choose a cell i at random
9:       Compute fitness f_i and fitness of adjacent cells
10:      Apply GA to cell i [crossover and mutation]
11:   end for
12: end while
```

3.6 Simple and Irregular: Random Boolean Networks

3.6.1 Introduction

An artificial neural network is an information processing system that is made up of a number of simple, highly interconnected processing elements — the *neurons* — which process information in parallel. As a simplification, the neuron might be considered as a sort of *detector* that detects the existence of some set of conditions and that responds with a signal that communicates the extent to which those conditions have been met [308].

One of the simplest neuron models is the boolean model. In the following, some relevant (and historically interesting) work around the boolean and the random boolean network models shall be mentioned.

Like Turing [408], already McCulloch and Pitts considered in 1943 boolean networks of very simple two-state neurons without the use of weighted connections or variable thresholds [261]. In 1956, Allanson published a paper on randomly connected neural networks [10]. The neurons he used were simple, but more complex than McCulloch-Pitts elements. He concluded the paper with the following statement:

> "The behaviour of the most elementary networks, assuming neurons to be far more simple than they are in reality, is more varied and complex than has normally been assumed" [10].

One of the earliest attempts to classify patterns by machine came from Olivier Selfridge [364, 365] in 1958. Courageous, he stated: "Can a machine think? The answer is certainly: yes". Selfridge contributed a significant model called *Pandemonium* for parallel processing, a model that was able to learn to classify patterns.

Large random logical nets made up from McCulloch-Pitts formal threshold neurons have also been investigated in 1962 by Smith and Davidson [381] and by Asbhy et al. [16].

Rozonoér [348] published a paper on random logical nets in 1968 (originally written in Russian). He analyzed the properties of logical random nets consisting of elements whose properties depend on parameters chosen at random. The connections among the elements were also chosen at random. Rozonoér suggested that:

> "[...] objects of this type may present a certain interest in connection with physiological models and, possibly, will have direct technical applications in the future" [348].

In 1971, Amari [11] published a paper on the characteristics of randomly connected threshold-element networks with the intention of understanding

some aspects of information processing in nervous systems. He showed that two statistical parameters are sufficient to determine the characteristics of networks. In his 1972 paper [12], Amari further investigated the stability of state transitions in logical nets of threshold elements. For example, he showed that a net reaches an equilibrium state within k state transitions if its initial state is located within a distance of the k^{th} stability number from the equilibrium state (in the sense of the Hamming distance).

Aleksander et al. [9] have developed a pattern recognition system—called *WISARD*—based on a network without feedback made up of boolean processing elements, that was later further developed [8]. The WISARD machine consists of a set of n-input boolean functions whose outputs are summed up. The key observation is that a boolean function may be implemented as a look-up table in RAM. Aleksander also investigated the stability of randomly interconnected boolean networks [7].

Fukushima's *Cognitron* [147] and *Neocognitron* [148, 149] laid another milestone for systems based on threshold units. Compared to the WISARD system, Fukushima used feedback connections in his networks.

Martland [252] showed that it is possible to predict the activity of a boolean network with randomly connected inputs, if the characteristics of the boolean neurons can be described probabilistically. In a second paper [251], Martland illustrated how the boolean networks are used to store and retrieve patterns and even pattern sequences auto-associatively. He used networks with feedback connections and neurons similar to Aleksander's WISARD neurons. The synchronously operating networks can work in two modes: (1) running mode and (2) training mode. Martland contributed further work about the behaviour of boolean networks: [253, 254].

Crayton C. Walker was one of the first persons to investigate the effect of the system size on the behaviour of complex systems, namely networks made up from elements that compute recursive logical functions of two binary inputs and two internal states [448]. Later, he also investigated attractors of sparsely connected boolean networks [449].

Very little work has been done around asynchronous random networks of threshold elements. Harvey and Bossomaier [182] have shown that they behave radically different from the deterministic synchronous version. Earlier, Grondin et al. investigated the asynchronous behaviour of threshold-element networks and the role of deterministic chaos [165]. More recent work about rhythmic and non-rhythmic attractors in asynchronous random boolean networks comes from Di Paolo [311, 312].

Important contributions around random boolean networks, their characteristics and dynamics came from Stuart Kauffman [212–215] (see also next section) and Weisbuch [454, 455].

In the domain of language acquisition, James Hurford published a paper entitled "Random Boolean Nets and Features of Language" [202]. He uses the framework of Kauffman's random boolean networks to cast several

properties of natural languages: complexity, interconnectedness, stability, diversity, and undeterminedness. Thereby, a given language is modeled as an attractor of the network.

The Ph.D. thesis of E. Mayoraz investigates feedforward boolean neural networks with discrete weights [258]. Different learning algorithms and the computational power is investigated.

For hardware implementations of boolean neural networks and simulated annealing optimization, one might take a look at Niittylahti [292–295].

3.6.2 Stuart Kauffman and Co.

Networks built of many interacting units, i.e., nodes and neurons, are used to study complex dynamical systems in may areas. *Random boolean networks (RBN)* form a class of networks in which the links between nodes and the boolean functions are specified at random. Stuart Kauffman stated:

> "I now introduce a formal model of rugged fitness landscapes, called NK model. In this model, N refers to the number of parts of a system—genes in a genome, amino acids in a protein, or otherwise. Each part makes a fitness contribution which depends upon that part and upon K other parts among the N. That is, K reflects how richly cross-coupled the system is. In the geneticist's term, K measures the richness of epistatic interactions among the components of a system" [213, p. 40].

Random boolean networks are usually specified by two parameters: N, the number of nodes and K, the number of incoming links per node (sometimes, K indicates the average number of links). The neighborhood $\Pi(x_i)$ of a node x_i is defined as

$$\Pi(x_i) \subset P_K(\{x_1, \ldots, x_N\} \setminus \{x_i\}) \tag{3.3}$$

where $P_K(X)$ denotes the set of all subsets of size K from X. The neighborhood size is given by $K = |\Pi(x_i)|$. The most common ways to chose the neighborhood are the following:

- *random neighborhood*: chose the K variables randomly from the set $\{x_1, \ldots, x_N\} \setminus \{x_i\}$ or from the set $\{x_1, \ldots, x_N\}$
- *adjacent neighborhood*: K variables are chosen in the immediate neighborhood.

$A(N, K)$ is sometimes used to denote a NK model with adjacent neighborhood, $N(N, K)$ for the random neighborhood. It is also easily possible to establish the interaction graph $G(V, E)$ where $V = \{x_1, \ldots, n_N\}$ corresponds to the set of variables or nodes, and $\{x_i, x_j\} \in E$ if and only if x_i is

connected to x_j. Figure 3.8 shows a possible NK random boolean network representation ($N = 8, K = 3$).

Note that K can refer to the *exact* or to the *average* number of interactions. As Kaufmann writes, the NK model is very similar to the famous and well-studied class of models which arises in statistical physics, called *spin-glasses* [118]. Spin-glasses are disordered magnetic materials in which the orientation of magnetic dipoles may be either parallel or antiparallel in space.

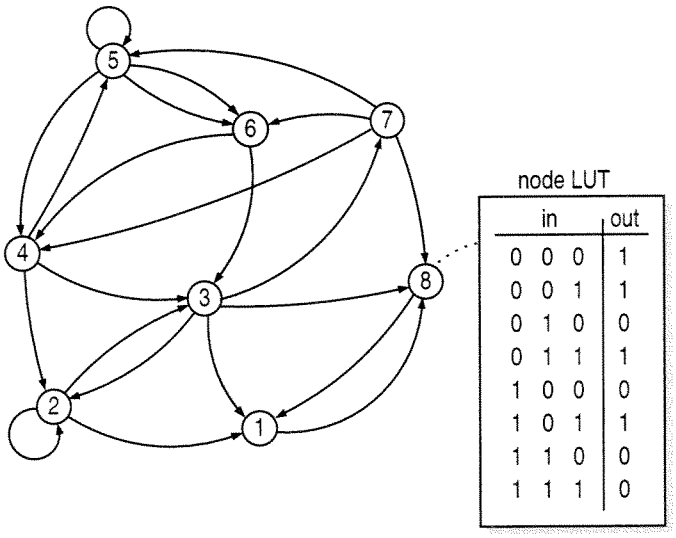

Figure 3.8: A NK random boolean network (RBN) with $N = 8$ and $K = 3$. The random boolean look-up-table of node 8 is provided as an example.

Synchronous random boolean networks have been seriously investigated by Kauffman [213], Weisbuch [455] and others. RBN have for example been used as models for biological phenomena such as genetic regulatory networks. Further important work about boolean network dynamics came from Martland [251, 252, 254], Amari [11, 12], Rozonoér [348], and Derrida and Pomeau [106]. Practical systems for pattern classification with boolean networks have mainly been developed by Igor Aleksander [7–9].

Very little work has been done around asynchronous random boolean networks (ARBN), although the updating scheme in discrete systems plays a crucial role for its properties. Moreover, for many physical and biological phenomena, the assumption of asynchrony seems more plausible. Harvey and Bossomaier [182] have shown that ARBNs behave radically different from the deterministic synchronous version. Earlier, Grondin et al. investigated the asynchronous behaviour of threshold-element networks and the role of deterministic chaos [165]. Di Paolo [311, 312] provided further analysis and mainly investigated rhythmic and non-rhythmic attractors. Al-

though ARBNs cannot exhibit strictly cyclic behavior (due to their random updating scheme), he has shown that they can all the same model rhythmic phenomenon. Recently, Gershenson [158] provided a first classification of the different types of RBNs:

- **CRBN – Classical Random Boolean Networks:** This is the "original" RBN update scheme proposed by Kauffman, where at each discrete time-step all nodes in the network are updated synchronously.
  Updating scheme: synchronous, deterministic.

- **ARBN – Asynchronous Random Boolean Networks:** At each time-step a single node is chosen at random and updated.
  Updating scheme: asynchronous, non-deterministic.

- **DARBN – Deterministic Asynchronous Random Boolean Networks:** Each node gets two parameters p and q, where $(p, q \in \mathbb{N}, q < p)$. At each discrete time-step k, all nodes that verify $q = k \mod p$ are updated sequentially (in an arbitrary order).
  Updating scheme: asynchronous, deterministic.

- **GARBN – Generalized Asynchronous Random Boolean Networks:** In this mode, a random number of nodes is now updated synchronously at each time-step.
  Updating scheme: semi-synchronous, non-deterministic.

- **DGARBN – Deterministic Generalized Asynchronous Random Boolean Networks:** Again, each node gets two parameters p and q where $(p, q \in \mathbb{N}, q < p)$. At each discrete time-step k, all nodes that verify $q = k \mod p$ are updated synchronously.
  Updating scheme: semi-synchronous, deterministic.

Gershenson's study also revealed that the RBNs point attractors are independent of the updating scheme and that they are more different depending on their determinism rather than depending on their synchronicity. Note, that for many physical and biological phenomena, the assumption of asynchrony seems more plausible. The goal of Stark and Hughes' paper [388], for example, was to demonstrate that irregular asynchronous automata nets — as opposed to cellular automata — are a realistic approach to modeling biological information processing.

Kauffman's studies have revealed surprisingly ordered structures in randomly constructed networks. In particular, the most highly organized behaviour (i.e., small attractors, small number of attractors, stable attractors, etc.) appeared to occur in networks where each node receives inputs from two other nodes ($K = 2$). It turned out that the networks exhibit three major regimes of behaviour:

1. *ordered* (solid),

2. *complex* (liquid), and

3. *chaotic* (gas).

The most complex and interesting dynamics correspond to the liquid interface, the boundary between order and chaos. In the ordered regime, little computation can occur. In the chaotic phase, dynamics are too disordered to be useful. The most important and dominant results of Kauffman's numerical simulations can be summarized as follows [213]:

- The expected median state cycle length is about $\sqrt{N}$ (where N is the number of network nodes).

- Most networks have short state cycles, while a few have very long ones.

- The number of state cycle attractors is about $\sqrt{N}$. Therefore, a random boolean network with 10,000 elements would be expected to have in the order of 100 alternative attractors. A system with 100,000 would have about 317 alternative asymptotic attractors.

- The most interesting dynamics appear with an average connectivity of $K = 2$ (the "edge of chaos").

- State cycles are inherently stable to most minimal transient perturbations.

- A perturbed state cycle can directly change only to a small number of other state cycle attractors in the system.

In his latest books [214, 215], Stuart Kauffman presents new investigations around random boolean networks. For example, he shows that random boolean networks in the ordered regime have a power law distribution of avalanches in gene activities produced by reversing the activity of a single randomly chosen gene. On the other hand, in the chaotic regime, in addition to the power law distribution of avalanches, a large spike of vast avalanches appears that affects $30 - 50\%$ of the genes. Furthermore, he shows that networks with more than $K = 2$ inputs per node can be shifted into the ordered regime from the chaotic regime by choosing appropriate, so called "canalizing" boolean functions.

For further details, the interested reader is referred to [213, 215].

3.7 The Synchronization Task for Random Boolean Networks

The synchronization task for two dimensional and regular arrangements has already been presented in Section 3.5. However, solving the synchronization task on a completely regular grid abstracts from the original firefly synchronization task in nature and it would thus be more interesting and biologically plausible to use a random graph. The present section shall briefly present the synchronization task for random boolean networks.

To avoid problems during crossover between rules having different lengths, the number of incoming links per node/cell was held constant. Figure 3.9 shows a random graph with four incoming links per node. Using Stuart Kauffman's terminology (see for example [213]), the graph is a $N = 16$, $K = 4$ random boolean network. In this context, K specifies the exact number of inputs per node.

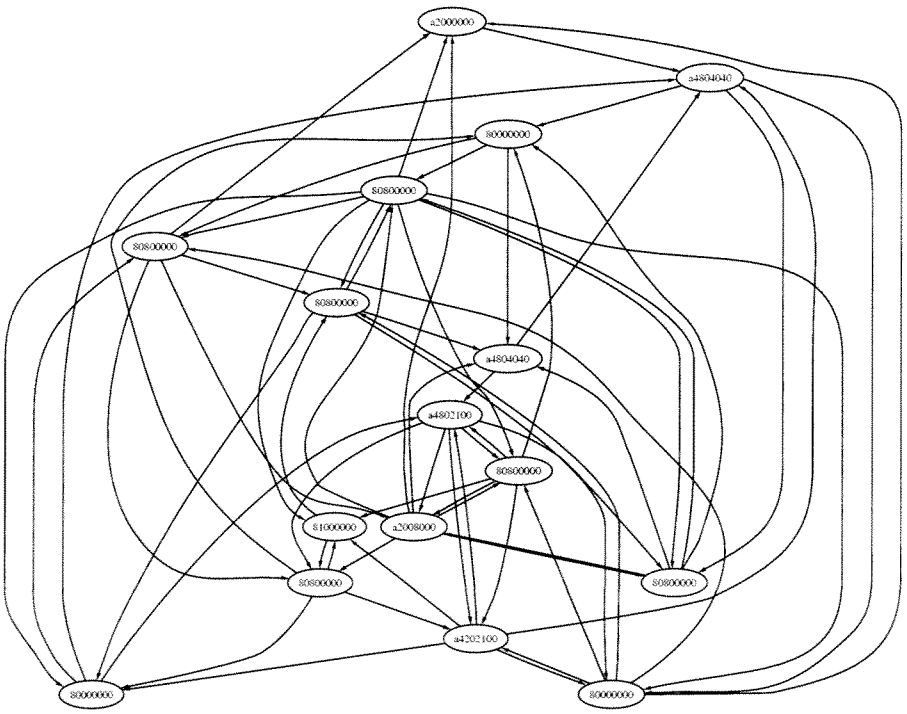

Figure 3.9: A random graph with 4 incoming links per node/cell. The eight different rules that allow to perfectly synchronize any initial configuration are written on the nodes (see Figure 3.3 for rule coding).

Using Algorithm 4 (Page 117), our co-evolutionary approach was able to find a perfect solution using eight different rules ($\nu = 0.859$, $H = 0.652$) for

the $K = 4$ random boolean network (RBN) of Figure 3.9. Figures 3.10 and 3.11 show the evolution of the number of rules for a $K = 4$ (on the left) and for a $K = 2$ RBN (on the right). It is astonishing that one rule is in general sufficient for the $K = 2$ network ($\nu = 0$, $H = 0$). Further investigations would be needed, but it almost certainly has something to do with Kauffman's findings that the most highly organized behavior appeared to occur in networks where each node receives inputs from two other nodes ($K = 2$). It turned out that Kauffman's networks exhibit three major regimes of behavior: *ordered* (solid), *complex* (liquid), and *chaotic* (gas). The most complex and interesting dynamics correspond to the liquid interface, the boundary between order and chaos. In the ordered regime, little computation can occur. In the chaotic phase, dynamics are too disordered to be useful.

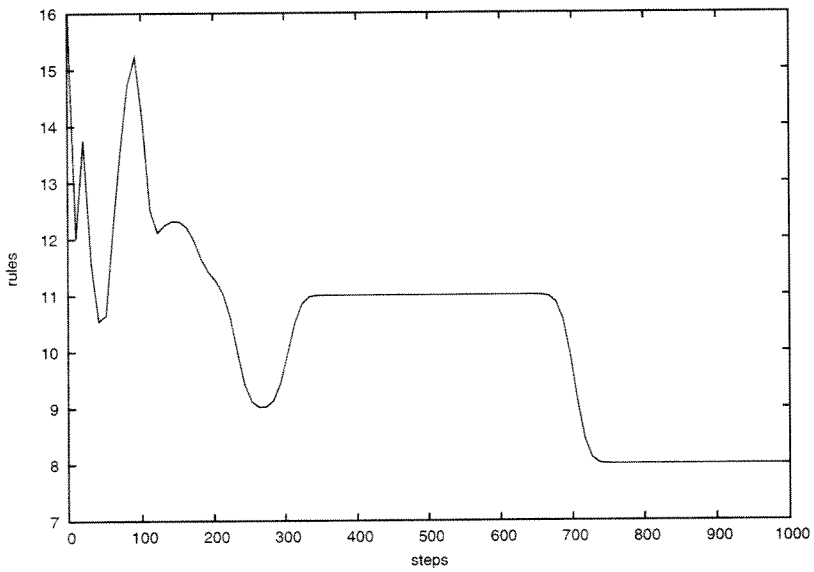

Figure 3.10: The number of different rules of a $K = 4$ random boolean network (RBN). Eight different rules are used.

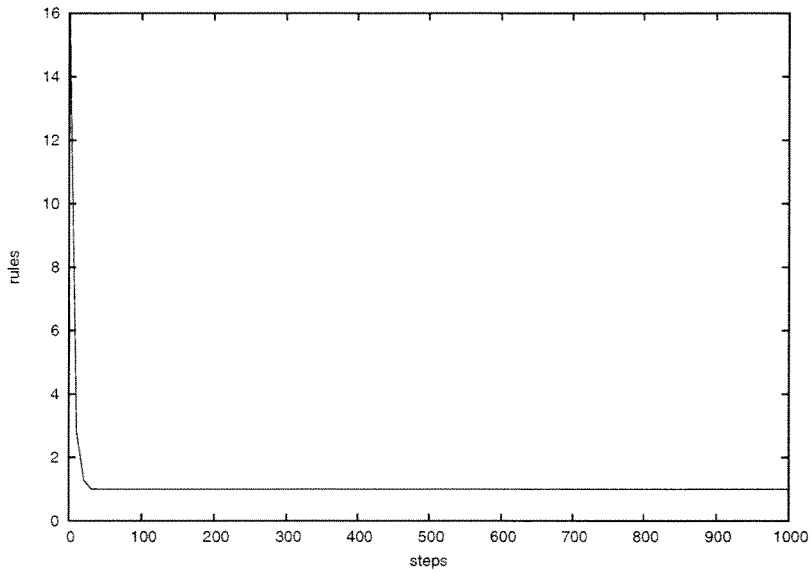

Figure 3.11: The number of different rules of a $K = 2$ random boolean network (RBN). Only one rule is used.

3.8 MATLAB Random Boolean Network Toolbox

In order to explore and simulate random boolean networks with various updating schemes (according to Gershenson [158]), a MATLAB random boolean network toolbox has been realized[4]. The toolbox is freely available from the following web-site:

```
http://www.teuscher.ch/rbntoolbox
```

The main goals was to provide a MATLAB toolbox for the simulation of synchronous and asynchronous RBNs. The toolbox not only provides all necessary functions to compute the evolution of the networks, but also offer the possibility to visualize the structure and the evolution of a RBN in the different updating modes. Furthermore, some more advanced functions are available, especially for topological evolution. A complete documentation with a function reference and some examples are also available in the web-site mentioned above. In the following, a short example shall illustrate some basics.

3.8.1 An Example

Cellular automata may be regarded as a special subclass of random Boolean networks. In a CA, a node only receives inputs from the nodes which lie in their purely local and homogeneous neighborhood. A CA can therfore well be simulated by our RBN toolbox.

Since the future state of each cell (node) only depends on the actual state and the states of its two adjacent cells, the connection matrix is a band-matrix with exactly three entries per column. The CA will be created in the form of a ring, i.e., first and the last cell being neighbors. This can be done with the toolbox as following:

MATLAB AMB toolbox example:

```
>> A = ones(90,90);
>> B = tril(A,-2);
>> C = triu(A, 2);
>> D = B + C;
>> D(1,90) = 0;
>> D(90,1) = 0;
>> D(find(D == 1)) = 2
>> D(find(D == 0)) = 1;
>> D(find(D == 2)) = 0;
>> conn = initConnections(D)
```

[4] I am indebted to Christian Schwarzer for the implementation.

The variable conn now contains the connection matrix for two-dimensional 90-cell CA. Next, the rules must be defined. For that example, each node shall contain the same rule (uniform CA), more precisely rule 165.

MATLAB AMB toolbox example:

```
>> for i=1:90
>> rules(:,i) = [1 0 1 0 0 1 0 1]';
>> end
>> rules = initRules(rules);
```

For our example, we want all cells to have their initial state set to 1, except for cell number 46, which is set to 0.

MATLAB AMB toolbox example:

```
>> initialStates = ones(90,1);
>> initialStates(46) = 0;
```

Creating the Node-Structure-Array and visualizing the topology
Then, the CA can be created and the topology visualized:

MATLAB AMB toolbox example:

```
>> node = initNodes(90,initialStates,zeros(90,1),...
                    zeros(90,1));
>> node = assocRules(node, rules)
>> node = assocNeighbours(node,conn);
>> node = displayTopology(node, conn, 'line');
```

Figure 3.12 shows the 90-cell CA topology, which has been arranged in the form of a ring.

In the next step, the CA will be evolved. Usually, each cell is updated synchronously with all other cells. This updating scheme corresponds to the CRBN scheme according to Gershenson's classification [158]. The following MATLAB function generates the typical images as shown in Figure 3.13

MATLAB AMB toolbox example:

```
>> [node, tsm] = displayEvolution(node, 90, 'CRBN');
```

3.8.2 Function Reference

For a complete (and clickable html) reference, see the web-site as mentioned above.

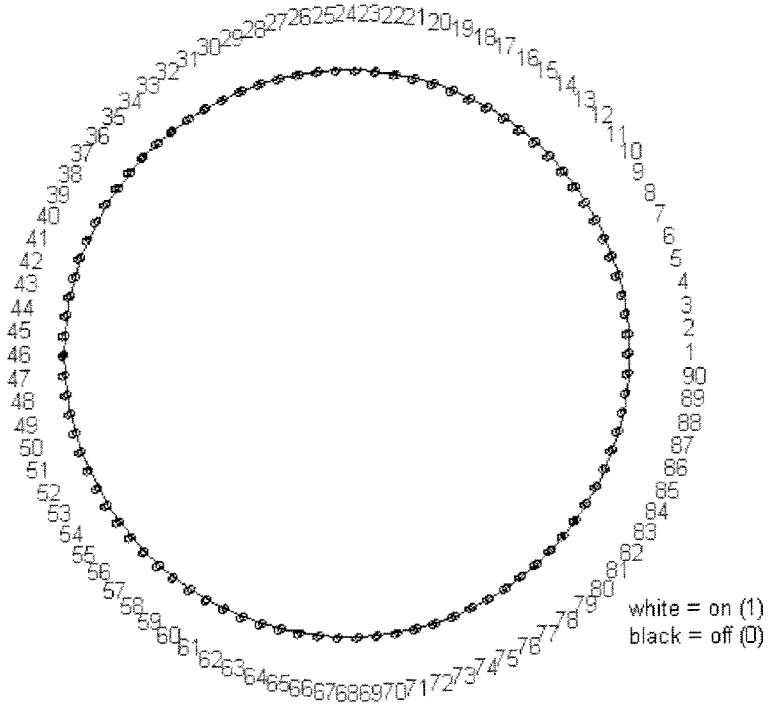

Figure 3.12: Visualization of the CA topology.

Initialization

```
initNodes()
initConnections()
initRules()
assocNeighbours()
assocRules()
displayTopology()
bsn()
bnKav()
arrow()
```

Evolution

```
evolveCRBN()
evolveARBN()
evolveDARBN()
evolveGARBN()
evolveDGARBN()
```

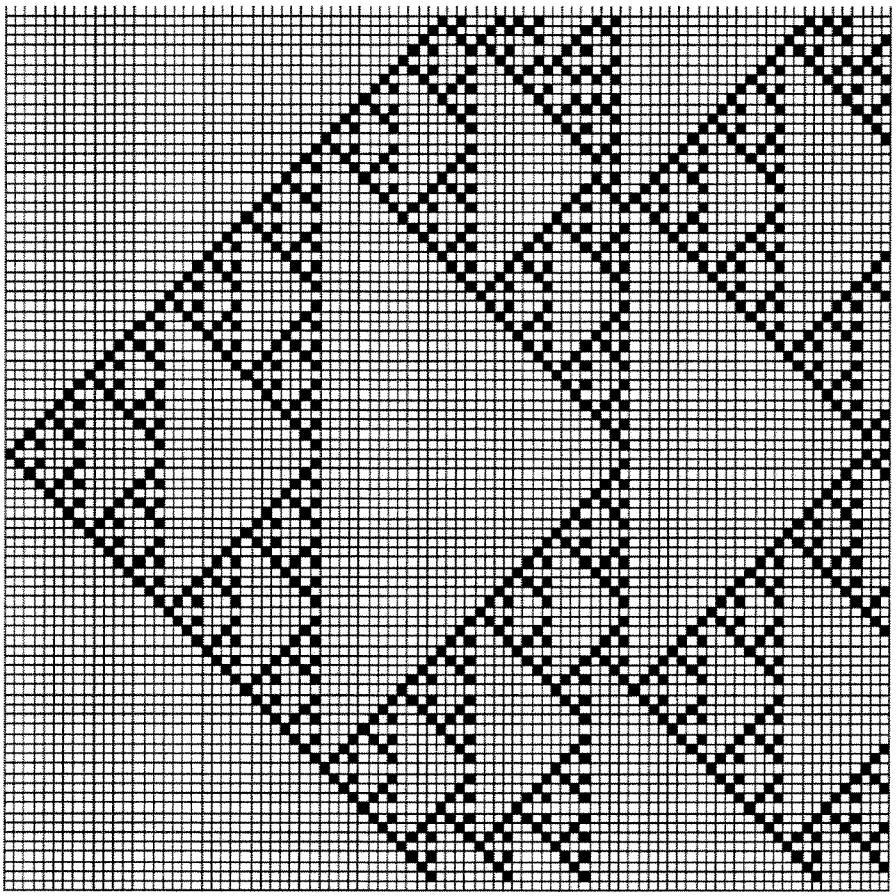

Figure 3.13: Visualization of the CA's time-state evolution.

```
displayTimeStateMatrix()
displayEvolution()
getStateVector()
setLUTLines()
setNodeNextState()
findAttractor()
evolveTopology()
frozenComp()
scalingLaw()
```

Statistics

```
resetNodeStats()
displayNodeStats()
countTransitionsPerNode()
```

```
averageConnectivity()
connectivityDistribution()
entropy()
```

Data Management

```
saveMatrix()
saveFigure()
```

3.9 Topological Evolution of Random Boolean Networks

In a recent paper [413], the topology evolution of Turing's unorganized (i.e., random) networks has been studied (see also Section 3.9). It has been shown, that a simple topology evolving algorithm for TBI-type networks and allowed the network to self-organize towards an average connectivity of about $K = 2$. For large systems ($N \to \infty$), the connectivity evolved towards the critical value $K_C = 2$. This result fully agrees with the results of Borholdt and Rohlf [48] who did the simulation with asymmetrically connected threshold networks: both models actually evolve towards the same critical connectivity $K_C = 2$ for large systems. As we have seen above, already Kauffman experimentally showed that the most interesting dynamics of RBN appear with an average connectivity of $K = 2$ (the boundary between order and chaos!) [213], however, he did not self-organize the topology of his networks.

The crucial question is "[...] whether a comparable mechanism may occur in natural complex systems" [48]. Bornholdt and Rohlf further state "[...] that this form of global evolution of a network structure towards criticality might be found in natural complex systems".

Of course, the finding that asymmetrically connected threshold networks as well as a certain type of Turing's unorganized machines evolve their topology towards the same critical value naturally arose the question whether this would also be valid for other networks and especially also for other updating schemes.

Note also that Sipper and Ruppin [375] performed simular experiments with cellular automata by evolving the cellular rules and the cellular connections to solve the density task. The connections per cell were however situated between $K = 5$ and $K = 7$.

Section 3.9.1 shall first briefly recall the results with Turing's unorganized networks, whereas Section 3.9.2 will provide some results with other updating schemes of random boolean networks.

3.9.1 Topological Evolution of Unorganized Machines

Topological evolution of dynamical networks has been studied in a recent paper by Bornholdt and Rohlf [48]. They evolved network topologies of an asymmetrically connected threshold network by a simple local rewiring rule: quiet nodes grow links, active nodes lose links. They have shown that this leads to an average connectivity of the networks towards the *critical* value $K_C = 2$ for large systems.

Kauffman postulated that gene regulatory networks may exhibit properties of complex dynamical networks near criticality [213], however, without providing an algorithm able to generate a topology near the critical point. Bornholdt and Rohlf asked "[...] whether connectivity may be driven towards a critical point by some dynamical mechanism" [48] and presented an approach that evolves the connectivity towards the critical point. In the remainder of this section we present a self-organizing topology evolving rule adapted for Turing's boolean networks that might also be applied to RBN.

The term *unorganized machines* (for a more detailed description, see also [408, 410]) has been defined by Turing in a rather inaccurate and informal way. An *unorganized machine* is a machine that is built up "[...] in a comparatively unsystematic and random way from some kind of standard components" [426, p. 9]. In terms of a digital system, the basic unit which Turing describes can be straightforwardly defined as an edge-triggered D flip-flop preceded by a two-input NAND gate. A positive-edge-triggered D flip-flop samples its D input and changes its Q output only at the rising edge of the controlling clock (CLK) signal. A global clock generator is used to synchronize the units.

An A-*type unorganized machine* is a machine built up from N units (neurons). Each unit receives inputs from exactly two other units. A second type of unorganized machine is called B-*type unorganized machine*. A B-type machine is an A-type machine where each connection between two nodes has been replaced by a small A-type machine, as shown in Figure 3.14(a). This small A-type machine functions as a sort of switch (modifier) that is either *opened* or *closed* (b). Turing simply wanted to create a switch based on the same basic element as the rest of the machine. The B-type link can be in three different states of operation: (1) it may invert the incoming signal (closed/enabled connection), (2) it may interrupt the incoming signal and put a constant 1 on its output (opened/disabled connection), or (3) it may act as in (1) and (2) alternately.

So far, the two unorganized machines we have described are, once initialized, no longer modifiable. It would clearly be interesting to modify the machine's interconnection switches (synapses) at runtime. Such a possibility would allow the use of an online learning algorithm that modifies the network's interconnections in order to train a net. Turing proposed to replace a B-type link by a connection as shown in Figure 3.14(c). Each interconnec-

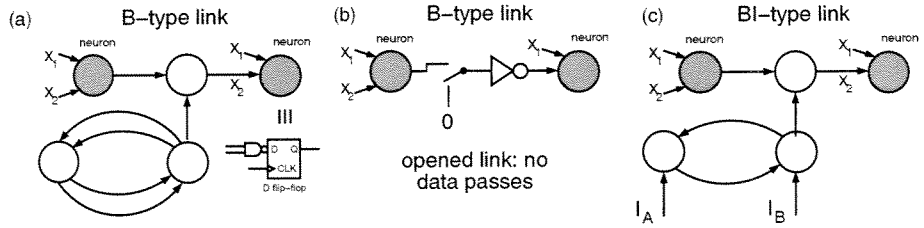

Figure 3.14: Turing's B-type links. Each B-type link is a small 3-node A-type machine. Part (b) shows a functional diagram of the switch incorporated in each node-to-node link: the example shows a disabled link). Part (c) shows a link with two interfering inputs that affect the internal state of the link.

tion has two additional *interfering inputs* I_A and I_B, i.e., inputs that affect the internal state of the link. By supplying appropriate signals at I_A and I_B we can set the interconnection into state (1) or (2). We will call this type of link a BI-type link (the I stands for "interference"). By means of this type of link, an external or internal agent can organize an initially random BI-type machine by disabling and enabling connections.

It is easy to see that—when abstracting from the one-clock delay of the D flip-flop—the principal computing element (one node) of an A-type network is the NAND operation since each node contains one 2-input NAND gate. It thus directly follows that every logical function can be computed by an A-type network since NAND units form a logical basis. On the other hand, one can see that a B-type node together with its two associated input links is nothing more that a simple OR gate—again, when abstracting from the one-clock delay of the D flip-flops—and thus not all boolean functions can be computed by a B-type machine. However, it is generally desirable to work with networks that offer universal computability. We therefore propose the TB-type link shown in Figure 3.15(a)—which is functionally equivalent to Copeland's and Proudfoot's proposal [85], but simpler. The additional node simply inverts the input signal. Figure 3.15(b) shows how two interfering inputs may be added to the introverted pair of the TB-type link. We call the resulting link TBI-*type link*.

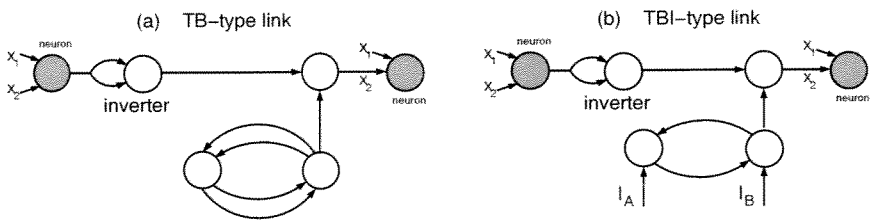

Figure 3.15: The TB-type (a) and TBI-type (b) link proposed by Teuscher. A simple node is used as an inverter in series with a normal introverted pair.

Modern neural networks are often organized in layers. For unorganized machines, the layered structure does not really exist since the machines are normally constructed at random and recurrent connections are allowed with no constraints.

As shown above, each link between two nodes of a TBI-type network is equipped with a small A-type machine that functions as a switch (see Figure 3.15). When the switch is opened, no information is passed, when it is closed, all information is passed and inverted. So far, all networks had an connectivity of exactly $K = 2$ connections per node. Algorithm 6 presents one of several possible iterating rules that evolves the average connectivity of TBI-type networks towards the critical point. Instead of creating and removing interconnections, the algorithm only operates the switches incorporated in each link and thus *configures* the network topology. The connectivity $K(i)$ of a given node i is equivalent to the number of incoming links that have closed switches. At the end of a successful topological evolution, links with opened switches might simply be removed.

Algorithm 6 Topological evolution of TBI-type networks

```
Choose a fully connected network (K = N)
Choose a random switch configuration with an average connectivity K_ini
for t = 1 to T time steps do
  Choose a random initial state vector s(0)
  Find a dynamical attractor and determine its length L
  for l = 1 to L do
    Run the network and record the activity A(i) of each node
  end for
  for all nodes i do
    if A(i) = 0 then
      A switch on link l_ji (where node j is chosen a random) is enabled/closed.
    else
      A switch on link l_ji (where node j is chosen a random) is disabled/opened.
    end if
  end for
end for
```

The activity $A(i)$ of a node i is defined as the number of changes it makes in the time interval $T_2 - T_1$. When algorithm 6 is applied to a TBI-type network, a typical picture as shown in Figure 3.16 arises. The network self-organizes towards an average connectivity of about $K = 3.7$ ($N = 30$ nodes, 10^4 time steps), independent of the initial connectivity (switch configuration). For the simulations we ran, the average connec-

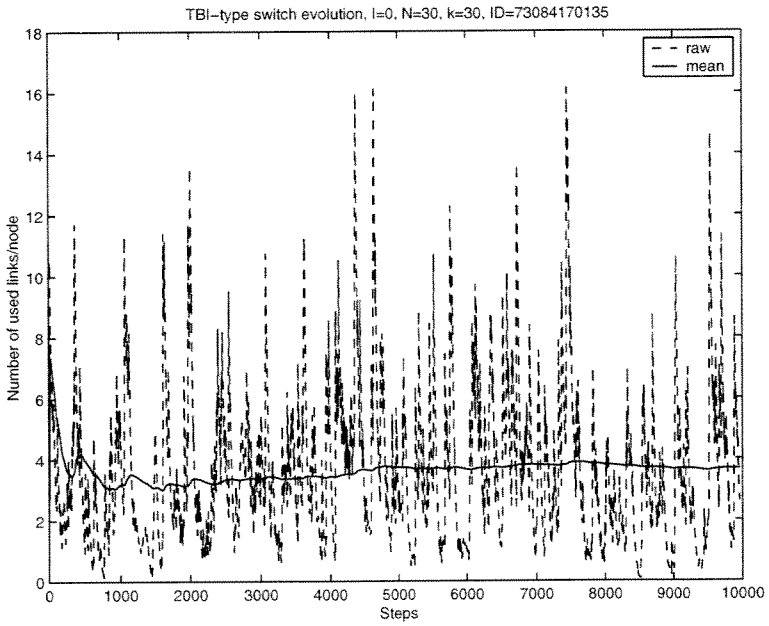

Figure 3.16: Evolution of the average connectivity of a TBI-type $N = 30$ nodes network. Self-organization started at $K = 20$ (all connection switches closed).

tivity approximatively obeys the scaling law presented by Bornholdt and Rohlf [48]: $K_{ev}(N) - 2 = cN^{-\delta}$ where $c = 12.4$ and $\delta = 0.47$. Thus, when $N \to \infty$, the network evolves towards the critical connectivity $K_C = 2$. Figure 3.17 ($N = 20$ nodes, 10^3 time steps) shows the evolution of the average attractor length and the average number of attractors when algorithm 6 is applied to a network. It can be seen that both values essentially remain constant.

The above topological evolution algorithm (algorithm 6) presents a new and different type of mechanism compared to the phenomenon of self-organized criticality [24]. The networks self-organize towards criticality and "[...] exhibit considerable robustness against noise in the system" [48]. Bornholdt and Rohlf have shown that the mechanism is based on a topological phase transition in the networks.

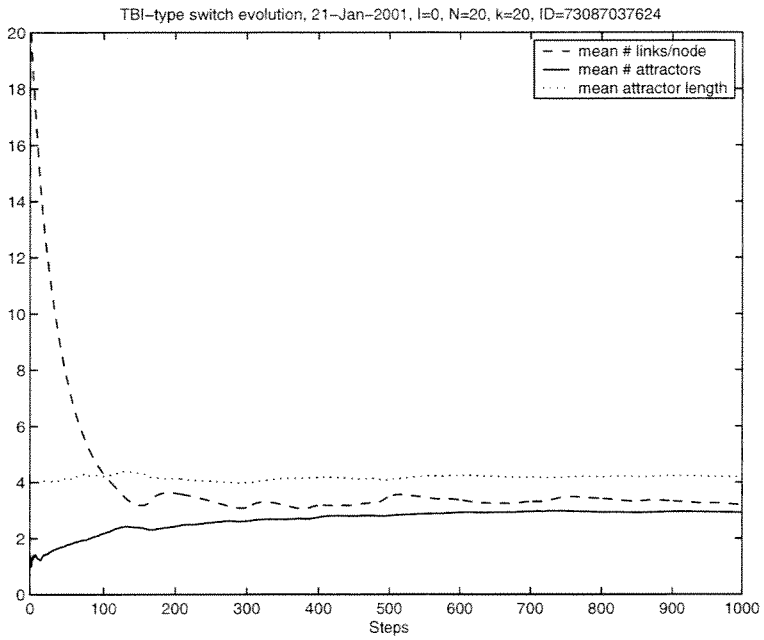

Figure 3.17: Typical average attractor length and average number of attractors during TBI-type topology evolution. Self-organization started at $K = 20$ (all connection switches closed).

3.9.2 Experiments with Gershenson's Updating Schemes

In the last Section, we have seen that a modified version of Bornholdt and Rohlf's topological evolving algorithm allowed to evolve the topology of Turing's unorganized machines to a critical value.

Topological Evolution

In order to investigate whether the same method might also be used for classical random boolean networks with various node updating schemes, I made several experiments with MATLAB toolbox as described in Section 3.8.

Figures 3.18 - 3.22 show the evolution of the average connectivity for the five different node updating schemes as presented in Section 3.6.2. The network size was $N = 15$ nodes and the initial connectivity was $K = 15$ and $K = 0$.

One can see, that all five plots evolve towards critical values, independently of the initial conditions and independently of the node updating scheme. Interestingly, the critical values of the ARBN and the CRBN experiments are very similar, whereas the DGARBN network has a fairly low critical value.

For all the experiments the evolutionary time t was discrete and at each time step, the activity of the network was measured over $L = 50$ time steps. The algorithm used is reproduced in the previous section.

In order not to get the network stuck in a local maximum, an additional parameter has been introduced which allows to randomly add or remove a certain number of connections, independently of the activity of the node. For all experiences, this parameter has been set to 1, i.e., at each time step one connection was randomly added or removed.

The following MATLAB code shows how the random boolean network MATLAB toolbox was invoked for the simulations. The first command creates the rule matrix and the structure of the network. The second command evolves the topology staring from a fully connected network, i.e., $K = 15$, whereas the third command starts with an unconnected network. Both data is then displayed in the same plot.

MATLAB AMB toolbox example:

```
>> [node, conn, rules] = bsn(15, 15, 'line')
>> [nodeUpdated, fHandleOut] = ...
     evolveTopology(node, 'CRBN', 1000, 15, 1, 50, 0)
>> [nodeUpdated, fHandleOut] = ...
     evolveTopology(node, 'CRBN', 1000, 0, 1,...
     fHandleOut, 50, 0, 50)
```

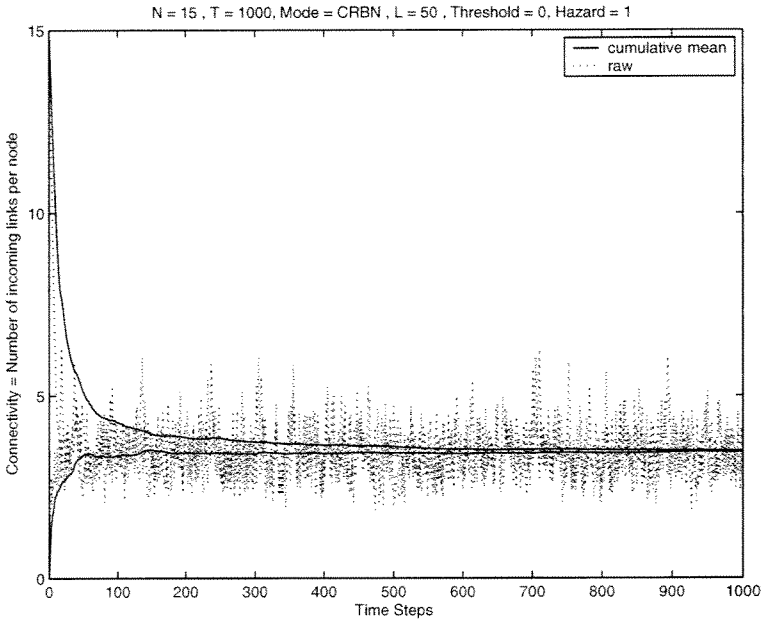

Figure 3.18: Evolving the topology of a random boolean network with "CRBN" updating scheme.

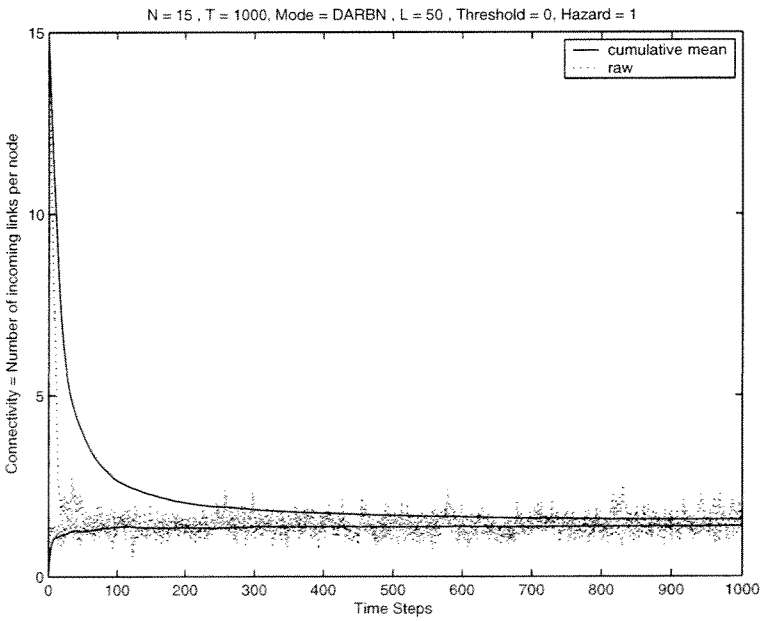

Figure 3.19: Evolving the topology of a random boolean network with "DARBN" updating scheme.

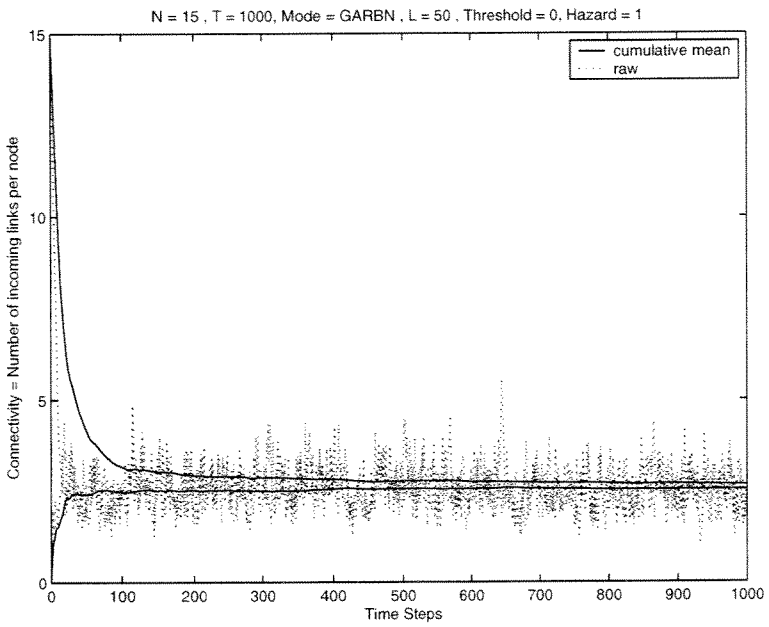

Figure 3.20: Evolving the topology of a random boolean network with "GARBN" updating scheme.

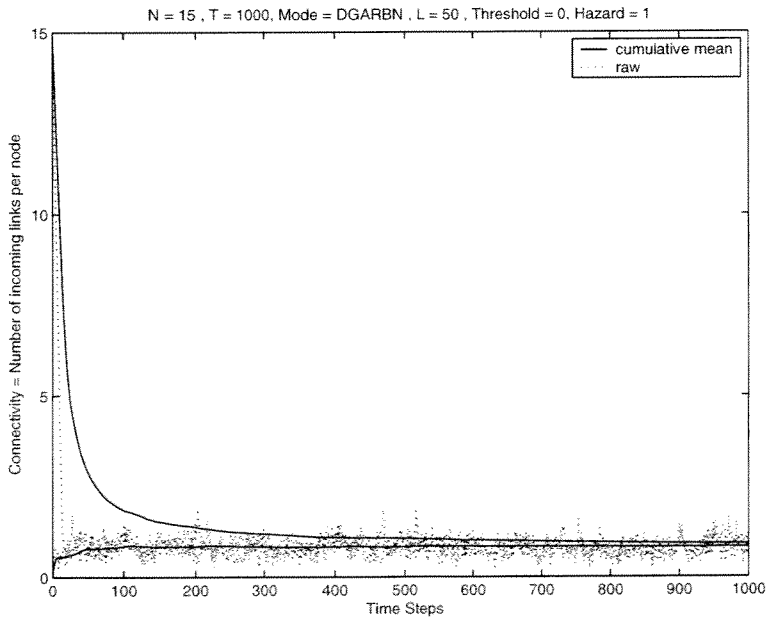

Figure 3.21: Evolving the topology of a random boolean network with "DGARBN" updating scheme. The critical value is fairly low compared to the other networks.

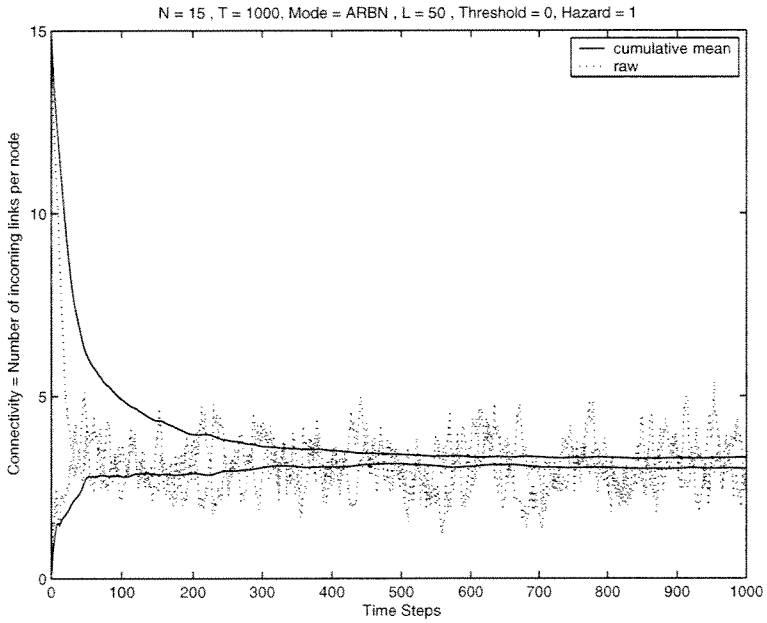

Figure 3.22: Evolving the topology of a random boolean network with "ARBN" updating scheme. Even in the asynchronous case, the network self-organizes towards a critical value.

The random boolean network MATLAB toolbox does unfortunately not allow to simulate large networks. This is the reason why I was unable to plot useful log-log plots of the network connectivity in function of the network size N and to determine the exact scaling law for each type of network.

The self-organization of the network's interconnectivity towards a critical value $K \to K_c$ presents a very interesting phenomenon. In particular, the algorithm is highly robust and independent of the initial conditions. As Bornholdt and Rohlf state, "[t]he main mechanism is based on a topological phase transition in dynamical networks" [48].

Frozen Component C(K,N)

In their original paper [48] Bornholdt and Rohlf also investigated the *frozen component* C(K,N) of their random threshold networks, which is closely related to the average activity of a network node. The *frozen component* C(K,N) is defined as the fraction of nodes that do *not* change their state along the attractor. In the limit of large N, C(K) undergoes a transition at K_c for classical random boolean networks.

The following MATLAB function shows the single random boolean network MATLAB toolbox command necessary to perform the frozen component

experiments:

MATLAB AMB toolbox example:

```
>> frozenComp(10, 15, 100, 0.2, 'CRBN')
```

Figures 3.23 - 3.27 show the frozen component plots for each of the five node updating schemes. The network size was set to $N = 15$ nodes and the simulations were conducted from $k = 0$ to a fully connected network, i.e., $k = 15$ in steps of 0.2 connections. Remember that K represents the average connectivity in this case. For each connectivity, the experiment was repeated $R = 100$ times and the average frozen component was then plotted. A frozen component of C(K,N)= 1 means that no node changes along the attractor were recorded, whereas a frozen component of 0 means that there are no stable nodes at all.

The phase transition for classical random boolean networks (CRBNs) in Figure 3.23 becomes very clear and corresponds to Bornholdt and Rohlf's findings in [48]. Note that the phase transition lies to closer to $K = 2$ the larger the network size N gets.

The plots are fairly different for DARBN, GARBN, and DGARBN networks. Although the frozen component decreases slowly, no obvious phase transition can be observed. In addition, the frozen component remains high, i.e., the networks are fairly stable.

The ARBN network presents an even more extreme case. The frozen component remains very close to 1 and there is no phase transition at all, which well corresponds to the findings of the next section.

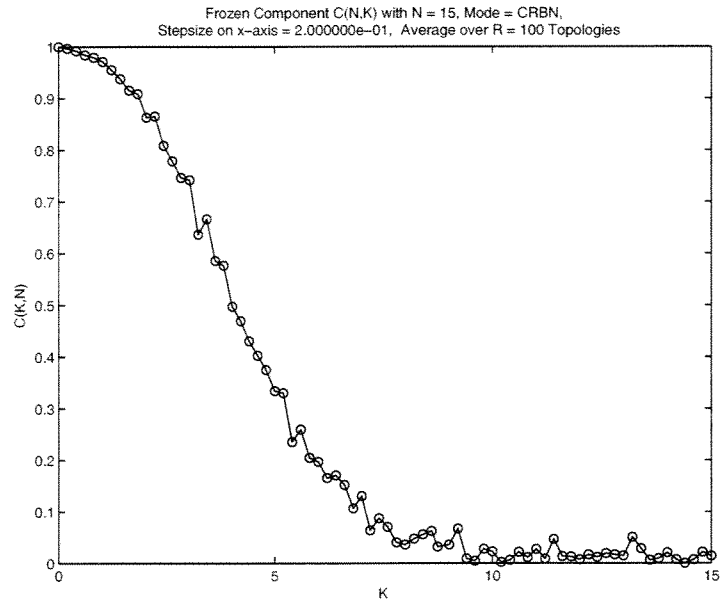

Figure 3.23: The frozen component C(K,N) of a random boolean network with a CRBN node updating scheme as a function of the networks average connectivity K.

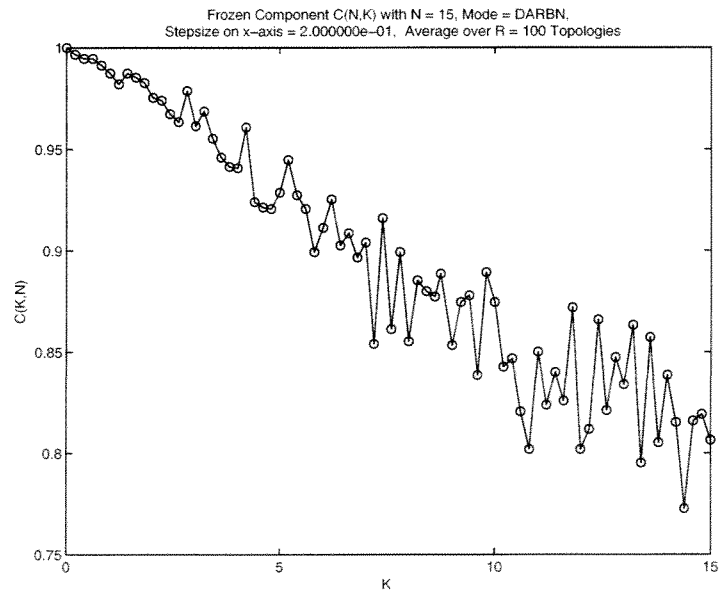

Figure 3.24: The frozen component C(K,N) of a random boolean network with a DARBN node updating scheme as a function of the networks average connectivity K.

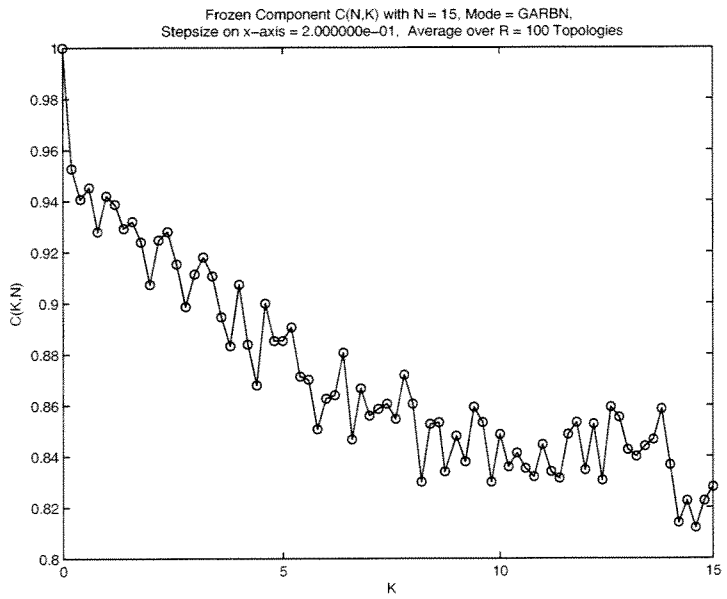

Figure 3.25: The frozen component C(K,N) of a random boolean network with a GARBN node updating scheme as a function of the networks average connectivity K.

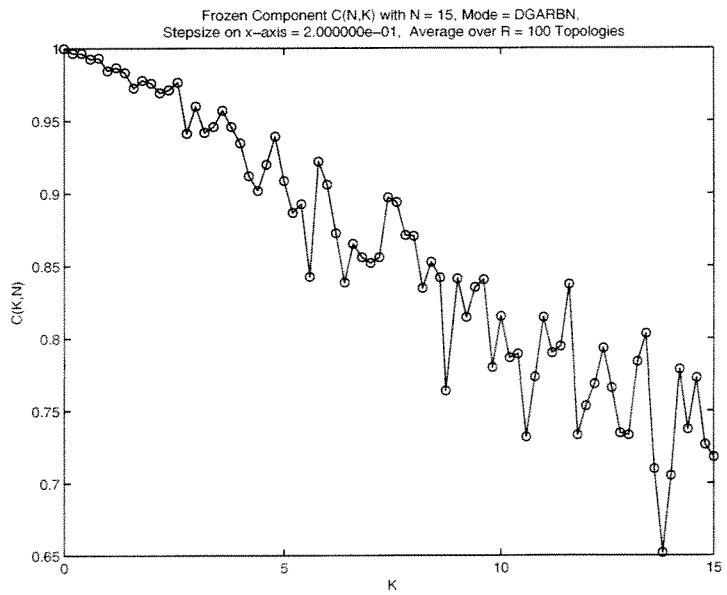

Figure 3.26: The frozen component C(K,N) of a random boolean network with a DGARBN node updating scheme as a function of the networks average connectivity K.

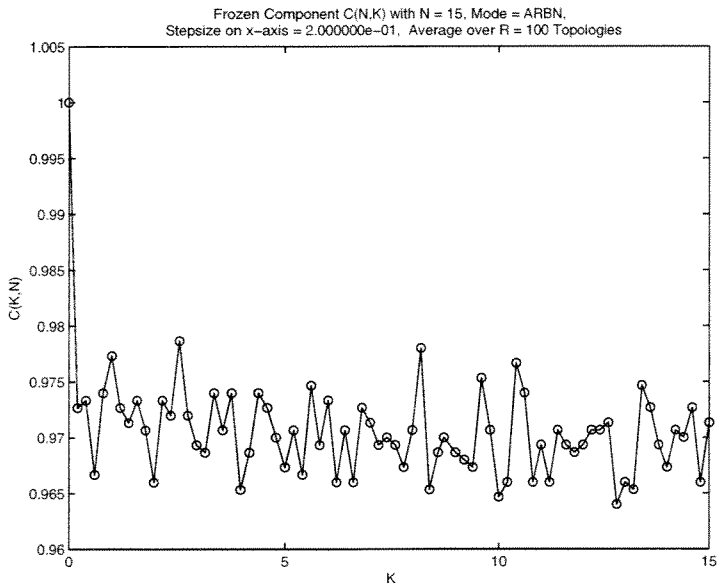

Figure 3.27: The frozen component C(K,N) of a random boolean network with a ARBN node updating scheme as a function of the networks average connectivity K.

3.10 Critical Values in Asynchronous Random Boolean Networks

As stated in Section 3.6.2, the average connectivity of $K = 2$ presents a critical connectivity for classical synchronous RBNs. This value is obtained by numerical simulations as well as by several theoretical methods.

This section presents work which has been accomplished together with Bertrand Mesot, to whom I am very grateful. The main results were also published in [264].

3.10.1 Derrida's Annealed Approach

In 1986, Derrida and Pomeau [106] proposed the *annealed approximation* which allowed to predict $K = 2$ as the critical value of K for synchronous random boolean networks. This section shall briefly recall the basic ideas of this approach, which will then applied in a similar manner in Section 3.10.2 to asynchronous random boolean networks.

Assume that we have a network made up of N nodes, each being randomly connected to K other nodes. Each node can be in one out of two possible states, 0 or 1, and the node's state after the next update is defined by a randomly chosen boolean function that takes the K incoming links as inputs. The *network state* at time t is defined as the vector of node states at time t, which can be written as $\mathbf{s}^t$.

The network states $\mathbf{s}^t$ for $t \geqslant 1$ are correlated to the network wiring and the node's boolean functions. Derrida noticed that this is somehow difficult to formalize, which lead him to the correct assumption that, since everything (i.e., wiring, transfer functions) is random in RBNs, randomly generating a new network after each update should not fundamentally affect the overall dynamics of the system. In order to find out the critical value for K, he compared the dynamics of two identically connected networks with state vectors $\mathbf{s}_1^t$ and $\mathbf{s}_2^t$. Let $\mathbf{s}_2^t$ be a perturbed (i.e., changing the state of a random number of nodes) copy of $\mathbf{s}_1^t$ and let us define d_t as being the *Hamming distance* between $\mathbf{s}_1^t$ and $\mathbf{s}_2^t$. The main question is then as follows: which value of K allows $d_t \to 0$ when $t \to +\infty$?

To answer this question, let a_t be the probability for a node to have the same value in both networks at time t. Let us then make two subsets of nodes called A and B. A contains all nodes that have equal states in $\mathbf{s}_1^t$ and in $\mathbf{s}_2^t$, B contains nodes with unequal states. Obviously, the probability for a node to belong to A at time t is a_t. Moreover, for each node two possibilities arise: (1) each node that is connected belongs to A, and (2) at least one of those connected nodes belongs to B. Hence, in the first case, the node's input will be the same in both networks since each connected

node belongs to A. This does not hold in the second case as at least one of those connected nodes belongs to B. For a given node, the probability of having all of its connections in A is therefore $(a_t)^K$, $\left(1-(a_t)^K\right)$ otherwise. Consequently, the node's probability of being in the same state in $\mathbf{s}_1^t$ as in $\mathbf{s}_2^t$ at time $t+1$ is 1 in the first case and $1/2$ in the second as it is supposed that states 0 and 1 are equally distributed. Indeed, if a node has the same input in both networks, it will surely have the same output and consequently be in the same state in both networks at the next time step. In the second case, inputs are different and outputs will thus be equal with probability $1/2$.

The evolution of a_t can therefore be described as a function of t by means of the following recursive equation:

$$a_{t+1} = (a_t)^K + \frac{1}{2}\left(1-(a_t)^K\right) = \frac{1+(a_t)^K}{2}.$$

By taking into account that $d_t = 1 - a_t$, one gets:

$$d_{t+1} = \frac{1-(1-d_t)^K}{2}$$

which describes the evolution of d_t as a function of time t. Figure 3.28 plots d_{t+1} as a function of d_t for $K = 1, 2, 3, 4, 10$. One can easily see that $K \in \{1, 2\}$ are the two only values that allow $d_t \to 0$ when $t \to +\infty$. Geometrically speaking, the plots for $K = 1$ and $K = 2$ lie below the identity function $d_t = d_{t+1}$ which implies that d_{t+1} tends toward 0 as time increases.

The results suggest that synchronous RBNs with a connectivity of $K = 2$ (the "edge between order and chaos") are particularly resistant to perturbations. This is mainly due to a phase transition in the number of frozen components within a network (see also [213]). Naturally, the question arises whether such a critical value exists in asynchronous RBNs. To the best of our knowledge, our attempt is the first one to investigate this question.

3.10.2 An Approach for Asynchronous Networks

Let us now consider the following case of asynchrony: at each time step, 1 to N nodes are randomly selected and synchronously updated.

As shown in the previous section, we consider two identically connected networks with different state vectors $\mathbf{s}_1^t$ and $\mathbf{s}_2^t$. Again, consider $\mathbf{s}_2^t$ as being a perturbed copy of $\mathbf{s}_1^t$. In the asynchronous case, three possibilities arise for each node: (1) the node is updated in both networks, similarly to synchronous RBNs; (2) the node is updated in only one of the networks; and (3) the node is not updated at all. One therefore needs to calculate the node's probability of being updated. Let m be the number of nodes updated at a given time t. Hence, the node's probability of being updated knowing m

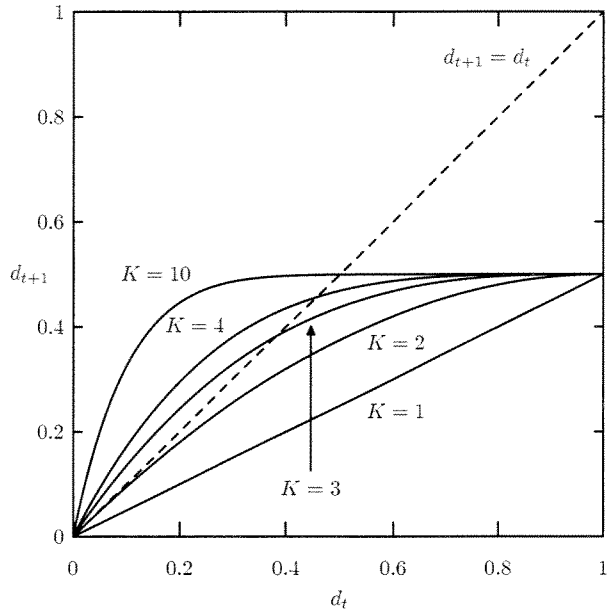

Figure 3.28: d_{t+1} in function of d_t for $K = 1, 2, 3, 4, 10$. For $K = 1$ and $K = 2$, d_{t+1} tends toward 0 as time increases. For more information, see also Kauffman [213].

is m/N. As $m \in [1 : N]$ and as it is supposed that all values are equally probable, the probability of being updated becomes:

$$P(\text{being updated}) = \frac{1}{N} \sum_{m=1}^{N} \frac{m}{N} = \frac{N+1}{2N} .$$

Given this, it is now possible to describe the probability of the three above-mentioned situations:

$$p_2 = \left(\frac{N+1}{2N}\right)^2 , \quad p_1 = 2\left(\frac{N+1}{2N}\right)\left(1 - \frac{N+1}{2N}\right) , \quad p_0 = \left(1 - \frac{N+1}{2N}\right)^2$$

where p_i describes the probability that the node is updated in i networks. Note that p_1 is counted twice because there are two possibilities of updating a node in only one of both networks.

Nodes can now again be separated into two subsets A and B with the same meaning as described in Section 3.10.1. Hence, if a node has all of its connections in A and is updated in both networks, it will surely hold the same value in $\mathbf{s}_1^{t+1}$ as in $\mathbf{s}_2^{t+1}$. If only one of both networks updates the node, there will be one chance out of two to be in a different state. Finally, if none of the networks updates the node, it will keep its current state at time $t + 1$ and thus hold the same value in both networks if and only if the

node belongs to subset A. After adding together all these probabilities one gets:

$$(a_t)^K \left[\frac{N+1}{2N} + \left(\frac{N-1}{2N} \right)^2 a_t \right]. \tag{3.4}$$

This represents the contribution of the nodes having all of their connected nodes in A to the overall probability a_{t+1} that a node has the same state in both networks at time $t+1$. To this term we must add the contribution of the nodes that have at least one connection in B. Note that when a node is updated in both networks, the probability of being in the same state at time $t+1$ is not 1 but 1/2, similarly to the synchronous case. We therefore obtain:

$$\left(1 - (a_t)^K\right) \left[\frac{3/2 \cdot N^2 + N - 1/2}{4N^2} + \left(\frac{N-1}{2N} \right)^2 a_t \right]. \tag{3.5}$$

And finally, the probability a_{t+1} that a node has the same state in both networks at time $t+1$ is obtained by adding 3.4 and 3.5 when $N \rightarrow +\infty$. Note that $N \rightarrow +\infty$ simplifies the writing of the forthcoming equations and is justified by the fact that, as long as we consider probabilities over nodes, a "sufficient" number of them is required. The resulting recursive equation for asynchronous RBNs is thus as follows:

$$a_{t+1} = \frac{1}{8} (a_t)^K + \frac{1}{4} a_t + \frac{3}{8}. \tag{3.6}$$

As for synchronous RBNs, this leads to the equation that defines the evolution of perturbations in asynchronous RBNs:

$$d_{t+1} = \frac{5}{8} - \frac{1}{8} (1 - d_t)^K - \frac{1}{4} (1 - d_t). \tag{3.7}$$

Figure 3.29 shows the plots of Equation 3.7 for $K = 1, 2, 3, 4, 10$. The following observations can be made:

1. Critical values for K do not seem to exist as no plot is strictly below the identity function $d_{t+1} = d_t$.

2. Each plot starts from one and the same point, situated at $d_{t+1} = 0.25$. This means that two networks with the same initial state will become different on 25% of their nodes at the next time step. Hence, $d_t = 0 \nRightarrow d_{t+1} = 0$ is no longer valid.

From the point of view of the above presented theoretical analysis, it follows that asynchronous RBNs do not seem to be as tolerant to perturbations as synchronous RBNs. In fact when d_t is lower than 0.5, d_{t+1} is always bigger than d_t, independently of K. This implies that, instead of reducing perturbations, asynchronous RBNs create new ones.

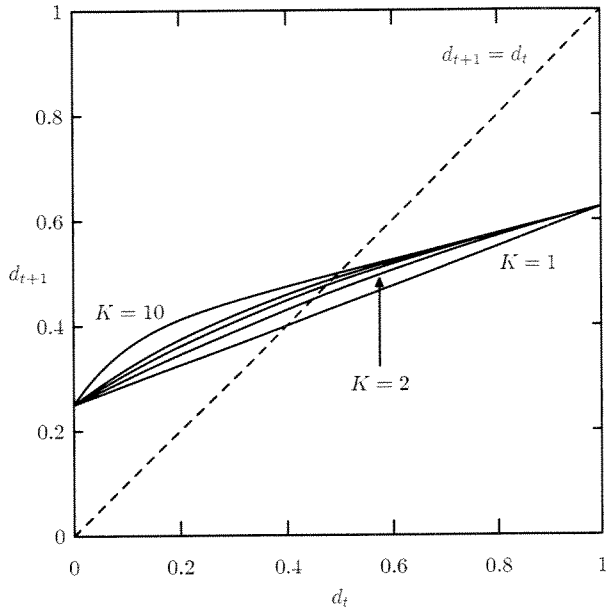

Figure 3.29: d_{t+1} in function of d_t for $K = 1, 2, 3, 4, 10$. Note that no critical value of K seems to exist.

Remember that Derrida allowed his networks to change at each time step. This simplification has proved to be correct for synchronous networks, i.e., not affecting the overall network dynamics, however, one might ask whether this hypothesis is correct for asynchronous RBNs too. The next section shall address this question by means of numerical simulations.

3.10.3 Numerical Results

Figure 3.30 shows numerical results for $K = 1, 2, 3, 4, 10$ obtained with networks of $N = 200$ nodes. For each value of d_t between 1 and 200, 200 randomly generated pairs of network states were tested during $t = 600$ time steps. Then, for each value of d_t, the mean value of d_{t+1} was computed.

Compared to Figure 3.29, Figure 3.30 shows mainly two differences. The plot obtained for $K = 1$ lies very close to the identity function $d_{t+1} = d_t$. Hence, the theoretical results do not correspond to the numerical simulations for that case. A possible explanation is that $K = 1$ networks are highly ordered ("solid") and that they possess a large number of short attractors. Indeed, two identical networks with their state vectors at distance d_t will very quickly end up into different attractors that are separated approximately by the same distance. Remember, however, that our theoretical model is rewired constantly and that therefore no attractors are possible, hence the difference between the simulations and reality.

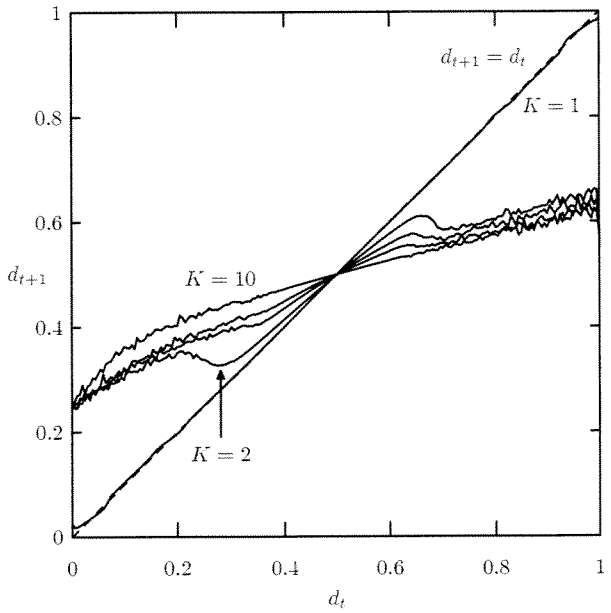

Figure 3.30: Numerical results obtained for $K = 1, 2, 3, 4, 10$ with asynchronous RBNs made up of 200 nodes.

The second important difference is that our theoretical model also partly fails to predict the behavior of the $K = 2$ plot for $d_t \in [0.2 : 0.7]$. The numerical results show that networks with two incoming connections per node can also be very stable within this interval as the plot lies rather close to the identity function.

Finally, the mean error between the theoretical model and the numerical simulations is about 3.3% for $K = 2$, 1.6% for $K = 3$ and falls below 1% for $K > 3$. Hence, the bigger K becomes, the better the model fits to reality.

3.10.4 Wrap-Up

The dynamic behavior of asynchronous RBNs has been investigated by means of a method inspired by Derrida's annealed approximation. The main question was whether there is a similar phase transition and critical value for K in asynchronous as in synchronous RBNs. The unique theoretical approach and numerical simulations revealed that asynchronous RBNs do not have a critical connectivity value similar to synchronous RBNs for K. Note, however, that Gershenson claims in recent article submitted to Physica D [157] to have found critical values in asynchronous random boolean networks. We were so far unable to reproduce his results but remarked an important difference in his and our approach: Derrida's approach requires the number of 1's and 0's in each rule to be equal. An analysis of Ger-

shenson's simulator revealed that this is not the case, which might explain some — but not all — of the differences in our results, but further detailed investigations will be necessary.

The asynchronous and nondeterministic updating scheme introduces perturbations that reach about 25% of the nodes and thus prevents the networks to become stable. Although there were some small discrepancies between the theoretical model and the numerical simulations, we can say that our approach to asynchronous RBNs predicts the most important characteristics of their overall dynamics. From the numerical simulations we can conclude that asynchronous RBNs tend to amplify small perturbations, to reduce big ones and to keep them constant when they are located in a region around $d_t = 0.5$.

CHAPTER 4

Biologically-Inspired Hardware

> ...they had built their hope of heaven on the binary system and the computer, 1 and 0, Yes and No...
>
> *The Armies of the Night*
> Norman Mailer, 1971

4.1 Introduction

Once a machine has been imagined and "built" on paper, an algorithm has been figured out in mind, the question whether and how to test the functioning of the idea naturally arises. This might be done by a simulation or by building the machine in reality. Of course, the usual way is to first simulate the system before a possible implementation is envisaged.

The question of whether to simulate systems in software or to implement them in (specialized) hardware is not at all a new one (see for example [155]). With the advent of *Field Programmable Gate Arrays (FPGAs)* [441], however, this question took somehow a back seat since it suddenly became easy and inexpensive to rapidly build (or rather *configure*) specialized hardware.

On the other hand, the question of how much "added value" a hardware implementation yields with regards to a simulation might and should of course be asked. It seems obvious that, as soon as we know the system's state tables and transition rules, we also know how to implement everything. This might be very complicated, of course, but we know that it is basically

an engineering problem "only". This is a first consideration, i.e., a physical realization of a system has in general no interest from the point of view of pure research.

A second consideration is the following: it is sometimes also argued that in order to create intelligent artefacts, we necessarily have to experiment with a physical realization ("embodied intelligence") since it is believed that intelligence arises (among other reasons) from an artefact's close interaction with its environment. I believe that to understand the mechanisms, we do not need realizations. After all, even the best realization is only a poor "simulation" of a real (biological or physical) system. The situation, however, is different if a hardware realization allows to notably speed up computations (since "computing in the space domain" is often more attractive than "computing in the time domain"), which is quite often the case, or if an implementation is required for validation and performance evaluation. From this point of view, hardware realizations will thus very probably continue to play a key role in the future.

In summary, software is always much more flexible than any physical realization and one should therefore always start with simulations first. Afterwards, a hard-headed and critical evaluation is required in order to determine the goals and pros and cons of a possible hardware realization.

In this chapter, I am going to present some relevant representatives of biologically-inspired hardware implementations.

4.2 The Embryonics Project

The *Embryonics* project (embryonic electronics) [245,246,248,310,403], is a project developed in our lab by Prof. Daniel Mange and his colleagues since 1993. It is inspired by the development of multicellular organisms and by the cellular division and differentiation of living organisms and their stem cells. The final goal of this approach was to develop extremely robust integrated circuits able to self-repair and to self-replicate. The new hardware paradigm, called *autonomous reconfigurable tissue*, is based on a homogeneous, infinitely expandable tissue structured in three layers: an input, an output, and a logic layer. A molecule—the tissue's basic element—consists of a reconfigurable digital circuit. A finite set of molecules makes up a cell, a small processor with an associated memory. The cell is capable of *autonomous self-replication* and can thus create the finite set of cells making up an organism, an application-specific multiprocessor system. The final organism can itself replicate, giving rise to a population of identical organisms, i.e., clones.

In the remainder of this section, the key-issues of the embyronics project are presented by means of an example.

4.2.1 Example: The BioWatch

In the framework of electronics, the environment in which our quasi-biological development occurs consists of a finite (but as large as desired) two-dimensional space of silicon. This space is divided into squares or *cells*. Since such cells (small digital processors) have an identical physical structure, i.e., an identical set of processing resources and of connections, the cellular array is homogeneous. Only the state of a cell, i.e., the contents of its registers, can differentiate it from its neighbors.

Our artificial organism, a bio-inspired watch called the *BioWatch*, is designed to count and display hours, minutes, and seconds, from 00h00'00" to 23h59'59". The input signal used for synchronizing the units of seconds is delivered by a wireless broadcast.

In the following, the main features of the BioWatch architecture and implementation are briefly discussed.

Multicellular Organization

The *multicellular organization* divides an artificial organism (*ORG*) into a finite number of cells (see figure 4.1), where each cell (*CELL*) realizes a unique function, corresponding to a modulo counting called *gene* of the cell. The same organism can contain multiple cells of the same kind (in the same way as a living being can contain a large number of cells with the same function: nervous cells, skin cells, liver cells, etc.). Moreover, each cell is associated with some *output state*.

The artificial BioWatch is a one dimensional organism with six cells and featuring four distinct genes ("mod 10" for counting the units of seconds or minutes, "mod 6" for counting the tens of seconds or minutes, "mod 10/4" for counting the units of hours depending on the value of the tens of hours, and "mod 3" for counting the tens of hours); the output state is the current value of the elapsed time and varies from 0 to 9 (for the units of seconds, minutes, and hours), from 0 to 5 (for the tens of seconds and minutes), and from 0 to 2 (for the tens of hours).

Cellular Differentiation

Let us call *operative genome (OG)* the set of all the genes of an artificial organism, where each gene is a unique function characterized by its position (its coordinates X and Y). Figure 4.2 shows the operative genome of the BioWatch, with the corresponding horizontal (X) coordinate; the vertical (Y) coordinate can be ignored in this particular case. Let then each cell contain the entire operative genome (figure 4.2): depending on its position in

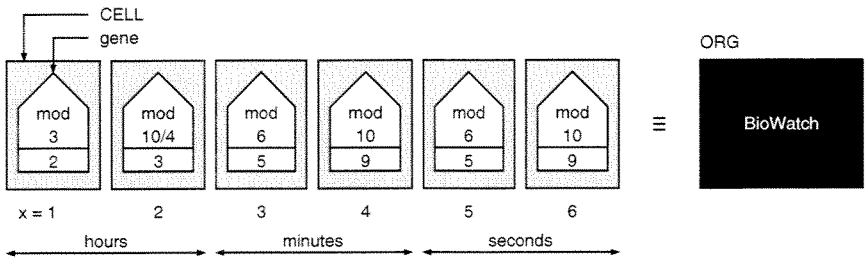

Figure 4.1: Multicellular organization of the BioWatch organism. In this example, the BioWatch displays 23h59'59".

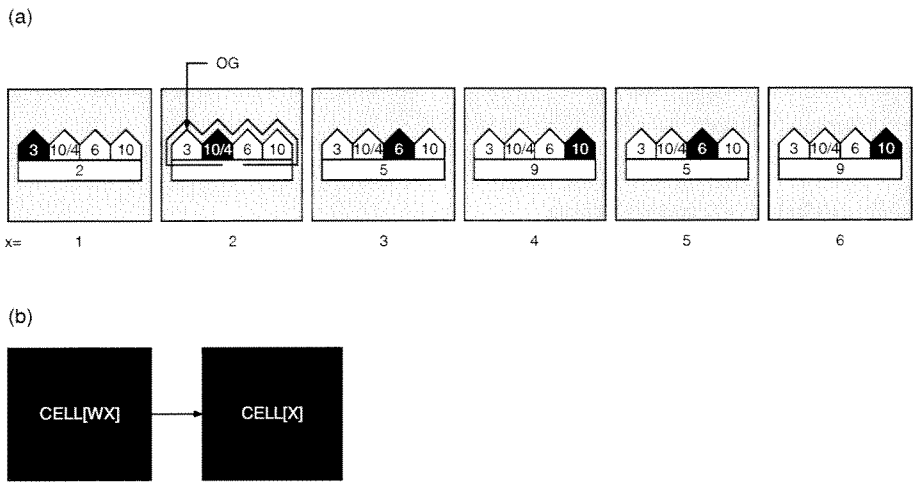

Figure 4.2: Cellular differentiation of the BioWatch organism with operative genome and coordinates. (a) Global organization; OG: operative genome (genes and coordinates). (b) Central cell $CELL[X]$ with its west neighbor $CELL[WX]$.

the array, i.e., its place in the organism, each cell can interpret the operative genome and extract and execute the gene which configures it.

In summary, storing the whole operative genome in each cell makes the cell universal: it can realize any gene of the operative genome, given the proper coordinates, and thus implement *cellular differentiation.*

In our artificial BioWatch, any cell $CELL[X]$ computes its coordinate X by incrementing the coordinate WX of its neighbor immediately to the west (figure 4.2). Any cell $CELL[OG, X]$ can thus be formally defined by a set of modulo-counting operations (its operative genome OG) and by its coordinate X.

Organism's Self-Repair

In order to implement *self-repair*, we need to add *spare cells* to the right of the original unidimensional organism (figure 4.3). These cells are defined by the coordinate $X = 7$.

The existence of a fault is detected by a $KILL$ signal which is produced in each cell by a built-in self-test mechanism. The state $KILL = 1$ identifies the faulty cell which is deactivated (column $X = 4$ in figure 4.3). All the functions (X coordinate and gene) of the cells at the right of the column $X = 3$ are shifted by one column to the right. Obviously, this process requires as many spare cells, to the right of the array, as there were faulty cells to repair (two spare cells tolerating two successive faulty cells in the unidimensional example of figure 4.3). It also implies that the organism has the capability of bypassing the faulty cell and to divert to the right all the required signals (such as the operative genome and the X coordinate, as well as the data busses).

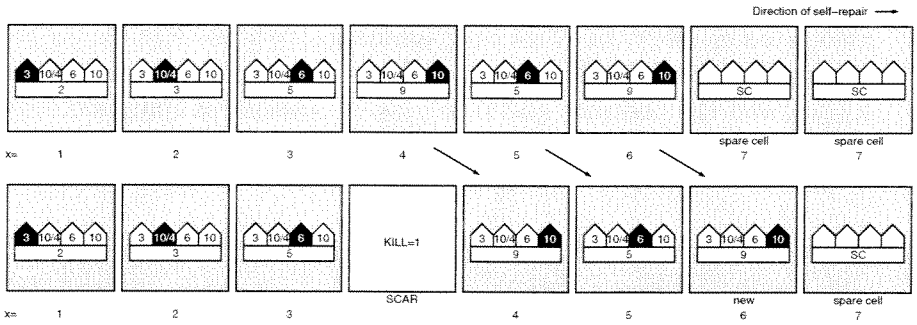

Figure 4.3: Self-repair of a 6-cell BioWatch organism with two spare cells (SC=spare cell) and one faulty cell.

The Cell's Architecture

In each cell of every living being the genome is translated sequentially by a chemical processor, the *ribosome*, to create the proteins needed for the organism's survival. The ribosome itself consists of molecules, whose description is an important part of the genome.

As mentioned previously, each cell is a small processor that sequentially executes the instructions of the artificial genome, the operative genome OG. The need to realize organisms of varying degrees of complexity has led us to design an artificial cell characterized by a *flexible architecture*, that is, itself configurable. It will therefore be implemented using a new kind of fine-grained field-programmable gate array (FPGA). Each element of our FPGA

(consisting essentially of a multiplexer associated with a programmable connection network) is the equivalent to a *molecule*, and an appropriate number of these artificial molecules allow us to realize application specific processors (cells).

4.2.2 The Embryonics Landscape

The final architecture of the Embryonics project is based on four hierarchical levels of organization which, described from the bottom up, are the following (figure 4.4):

- The basic primitive of our system is the *molecule*, the element of our new FPGA consisting essentially of a multiplexer associated with a programmable connection network. The multiplexer is duplicated to allow the detection of faults. The logic function of each molecule is defined by its molecular code or *MOLCODE*.
- A finite set of molecules makes up a cell, essentially a processor with an associated memory. In a first programming step of the FPGA, the *polymerase genome PG* defines the topology of the cell, that is, its width, height, and the presence and positions of columns of spare molecules. In a second step, the *ribosomic genome RG* defines the logic function of each molecule by assigning its molecular code or MOLCODE.
- A finite set of cells makes up an *organism*, an application-specific multiprocessor system. In a third and last programming step, the *operative genome OG* is copied into the memory of each cell to define the particular application, e.g., the BioWatch, executed by the organism.
- The organism can itself self-replicate, given rise to a *population* of identical organisms, the highest level of our hierarchy.

4.2.3 Wrap-Up

The Embryonics approach is certainly a very interesting concept, but unfortunately lacks several important links to reality. Although a prototype has been implemented as a VLSI chip, the embryonics architecture is not based on a realistic model of errors as they typically occur in silicon for real-world applications. In addition, the necessary cellular decomposition of an application as well as the fact that the entire program has to be stored within each cell pose major obstacles for real-world industrial applications. The price to pay for a fully scalable redundancy and for the universality is a loss in flexibility (due to the cellular decomposition as most applications are not easily decomposable into several parts) and an important increase of

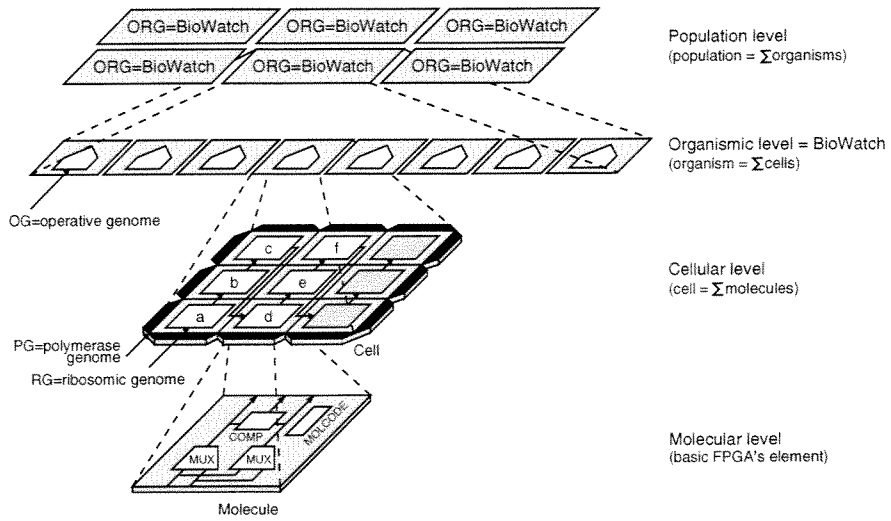

Figure 4.4: The Embryonics landscape of the BioWatch example: a four-level hierarchy.

the resources required (as the entire program has to be stored within each cell). There exists of course many alternative and somehow less "extreme" methods to fault-tolerance (see for example [186, 229, 244]).

George Church (professor of genetics at Harvard Medical School-MIT Division of Health Sciences and Technology, and director of the Lipper Center for Computational Genetic) said in the Wired news article "Computer, Heal Thyself"[1], which featured the Embryonics project, that he was not convinced the technology was completely groundbreaking in terms of computational self-repair. "The justification for including specialized hardware should be to actually achieve some level of hardware repair, but this does not seem to be what they have done. Reconfiguring around broken parts with redundant parts has been done for years in several commercial electronics applications including disk drives and charge-coupled devices (CCDs)", he further stated.

Fault-tolerance in digital circuits came up with the increased complexity of semiconductors and is now definitely a major concern to both system designers and users [231]. The fundamental question in estimating the system's reliability is whether it will function in a prescribed manner in a given environment for a given period of time. This, of course, depends of many factors. A reliability analysis of the Embryonics project has been done at the University of York [309, 310].

As presented in [231], there are numerous possibilities and levels where fault-tolerance can be applied in digital systems design. One thing is clear:

[1] `http://www.wired.com/news/technology/0,1282,53729,00.html`.

without such techniques, no digital memory or computer would correctly work today. For a recent survey of VLSI circuit reliability and fault classes see [78].

4.3 The BioWall

The main idea behind the construction of the BioWall[2] is the realization of Embryonic machines (see Section 4.2 and [389, 390, 404]). The structure of such machines, described in some detail below, is hierarchical: organisms (application-specific systems) are realized by the parallel operation of a number of cells (small processors), and each cell is implemented as an array of molecules (programmable logic elements).

4.3.1 The Technology Behind the Scenes

The BioWall is structured as a two-dimensional tissue composed of units (each unit corresponds to a molecule), where each unit (Figure 4.5) consists of an input element (a touch- sensitive membrane), an output element (an array of $8 \times 8 = 64$ two-color LEDs), and a programmable computing element (a Spartan XCS10XL Xilinx FPGA). The circuits are mounted on double boards (Figures 4.7, 4.8 and 4.9): the logic board contains $5 \times 5 = 25$ Xilinx FPGAs, while the display board is made up of the displays and the membranes. The two boards are rigidly bound together and connected by a bus to allow two-way communication between the logic and the display (a dedicated circuit on the logic board automatically distributes the signals to and from the displays).

[2] For up-to-date information see also `http://lslwww.epfl.ch/biowall`.

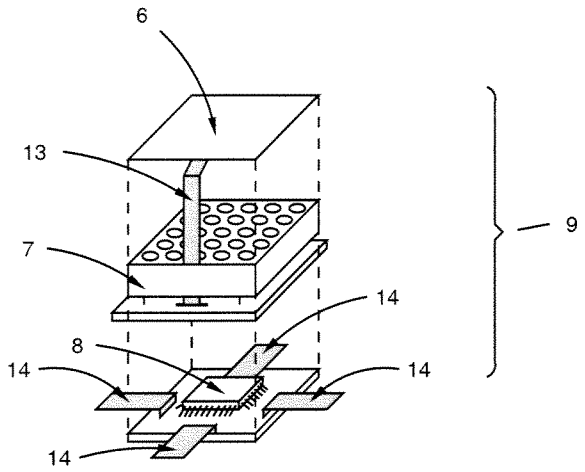

Figure 4.5: An illustration of the BioWall's basic building block (9). (6) touch-sensitive membrane, (7) bi-color 8×8-dot LED display, (8) Spartan XCS10XL Xilinx FPGA, (13) and (14) connections.

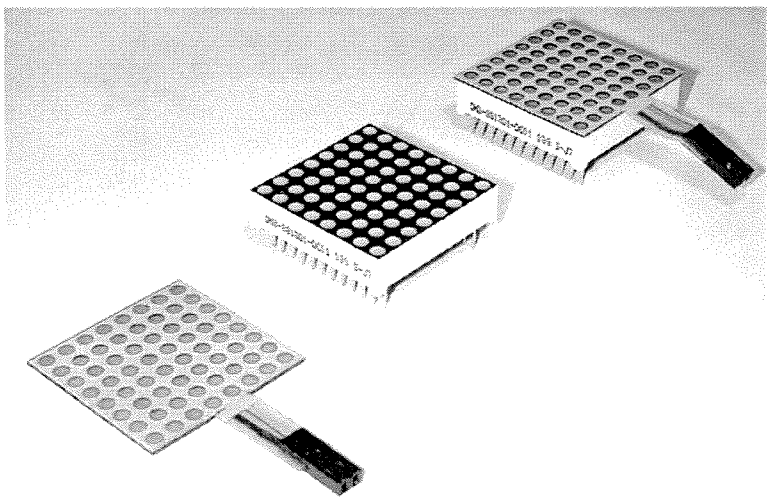

Figure 4.6: The molecule's touch sensitive membrane is glued on the 8×8-dot two color LED display. Photo: André Badertscher.

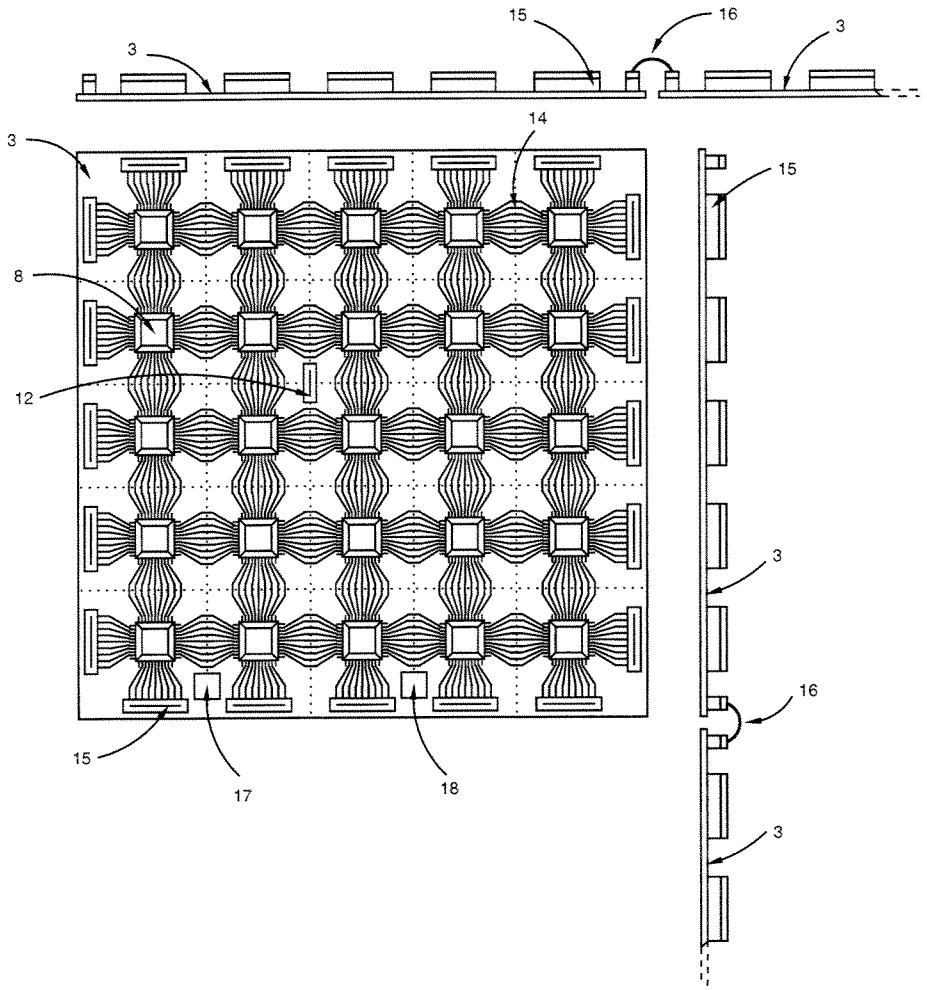

Figure 4.7: The printed circuit board (PCB) that assembles 5×5 basic elements. (8) Spartan XCS10XL Xilinx FPGA, (3) PCB, (15) connectors, (16) flexible connections, (17) and (18) power plugs.

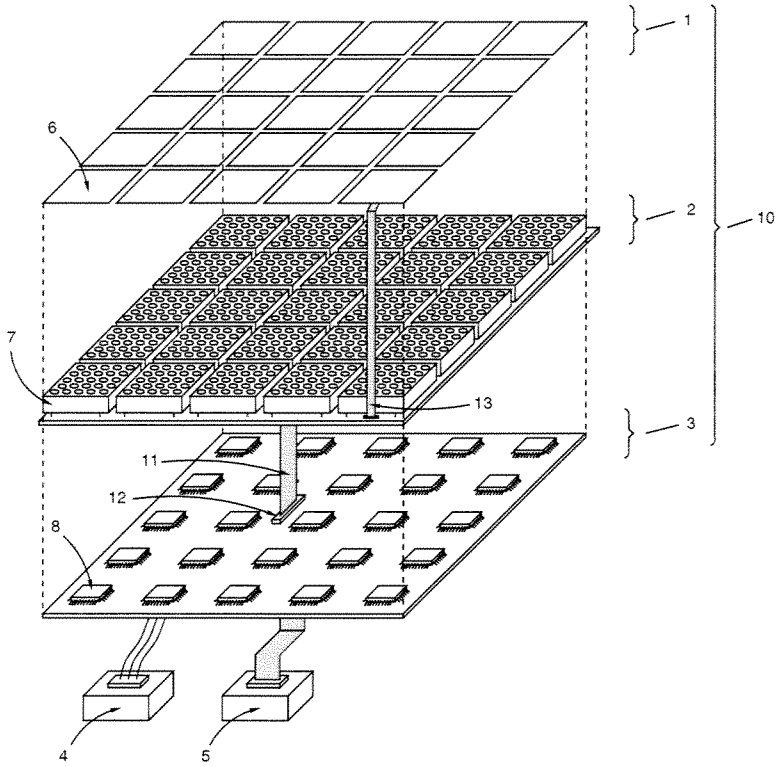

Figure 4.8: Assembly of a 5×5 basic elements module (10). (1) inputs, (2) outputs, (3) computational units, (6) touch-sensitive membrane, (7) bi-color 8×8-dot LED display, (8) Spartan XCS10XL Xilinx FPGA, (4) and (5) power supplys and external control unit, (12) connector, (13) and (14) connections.

On the logic board, the Spartans are placed in a regular two-dimensional grid (Figures 4.7, 4.8 and 4.9) and a subset of the pins of each FPGA (approximately 20 per side) are used to assure a direct pin-to-pin connection between each circuit and its four cardinal neighbors. The pins of the FPGAs placed along the edges are brought to a set of connectors to allow the pin-to-pin association to continue across boards (thus allowing the creation of perfectly uniform surfaces of FPGAs spanning as many boards as required). The remaining pins of each FPGA are connected to a centralized circuit that handles the distribution of the global signals (the clocks, the resets, and the configurations of the FPGAs) that arrive from the outside. We have built 228 such boards (not counting spare material), for a total of 5,700 units. The architecture of the boards implies that they can be seamlessly connected with each other to form a uniform surface of any shape and size (Figure 4.10). Figure 4.11 schematically illustrates the additional printed circuit boards and interconnection structure required to distribute the configuration bit-streams to each FPGA. One can also see that one

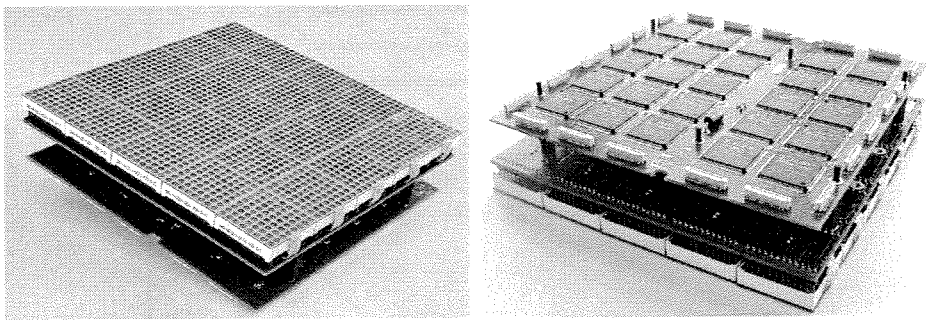

Figure 4.9: The 5×5-molecule module (front and rear view). Photo: André Badertscher.

power supply is used to supply ten FPGA printed circuit boards. On both lateral borders as well as at the bottom special printed circuit boards allow to interface the array of FPGAs.

Throughout the development phase and the lifetime of the machine, we have constructed several independent machines:

- $160 \times 20 = 3,200$-unit machine (Figure 4.13) was on display at the Villa Reuge museum;
- a $80 \times 25 = 2,000$-unit machine is kept in our laboratory to develop and test new applications;
- a $10 \times 15 = 150$-unit machine was embedded, together with the necessary control logic to charge applications, into a suitcase for portability;
- finally, we have built a $25 \times 160 = 4,000$-unit machine that was on display at the Telecom'03 conference in Geneva in October 2003.

The tissue represents then an impressive amount of computational power (up to 5,700 FPGAs), coupled with an I/O interface (the membranes and the LED arrays) that allows a large-scale visual and tactile interaction. The advantages of this solution are obvious: on one hand the size of the display allows an immediate interaction with applications that are normally limited to software simulation on a computer screen (some of these applications are described in the next section, others are mentioned in the conclusion), and on the other hand the computing power and programmability of the FPGAs allow the prototyping of new bio-inspired systems.

In the current version of the machine, the Xilinx FPGAs can only be programmed with the same configuration, which limits the functionality of the units to the 10,000 equivalent logic gates of the Spartans, while the considerable delays inherent in propagating a global signal over distances measured in meters seriously limit the clock speed (we have not tried to

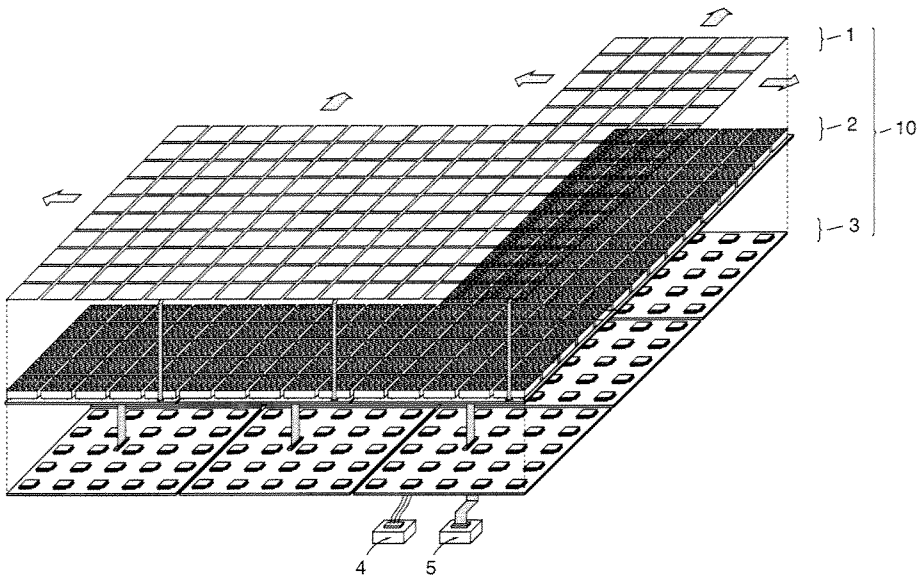

Figure 4.10: Global view of the reconfigurable tissue (10). An unlimited number of modules can be assembled. (1) inputs, (2) outputs, (3) computational units.

push the clock to its limits, as the current frequency of 1MHz is more than adequate if coupled with the massive parallelism of the machine and considerably too fast for human interaction in many applications).

Besides the I/O capabilities of the membranes and of the LED displays mentioned above, a set of modules placed on the borders of the machine allow the tissue to be interfaced with standard logic, either via a PC or directly with user-defined modules (the modules, of course, allow access only to the borders of the array, but, if necessary, signal propagation logic can be programmed in the FPGAs).

The software tools developed for the BioWall are rudimentary but complete. A simple interface on a PC allows the user to define a set of files that will be used to configure the tissue. Four kinds of files are currently defined (more can be added): the configuration file for the Xilinx FPGAs, and three different formats used to send user-defined data on the input pins at the borders of the tissue (used, for example, to provide an initial configuration for a cellular automaton). The values on the output pins at the borders of the tissue can be read by the PC and either stored on disk or used as required.

The BioWall was designed with a specific application in mind: the realization of ontogenetic machines as defined by the specifications of the Embryonics project. However, the capabilities of the BioWall are not limited to this application. Its cellular structure is well suited to the implementation

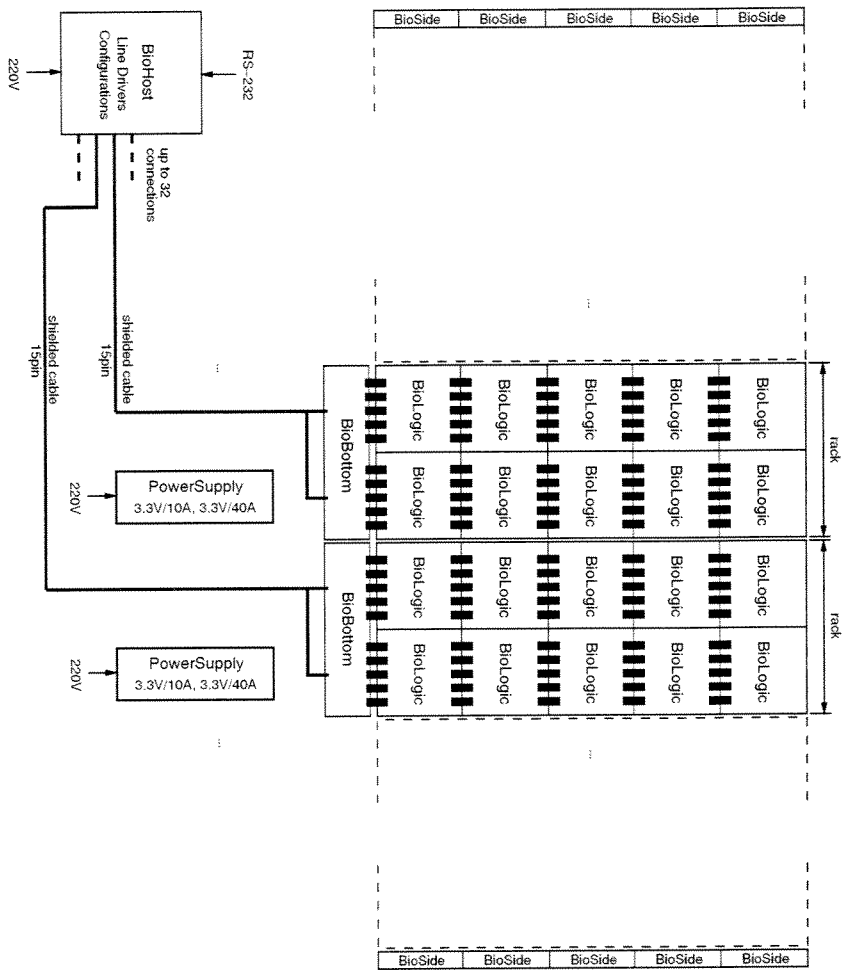

Figure 4.11: Schematic view of the additional printed circuit boards and interconnection structure required to distribute the configuration bit-streams to each FPGA.

of all sorts of bio-inspired applications. In this section, we will present a few such applications, to show how the BioWall can exploit the versatility inherent in its programmable logic and in its architecture to implement hardware inspired by all the three models of biological inspiration: phylogenesis, ontogenesis, and epigenesis.

4.3.2 The BioWatch on the BioWall

The principles of the Embryonics project as well as the theory behind the BioWatch have been described in detail in a number of publications (see also Section 4.2). In this section we will describe a large-scale implementation on the BioWall's computing substrate of the fault-tolerant and self-repairable BioWatch timer (as presented in Section 4.2).

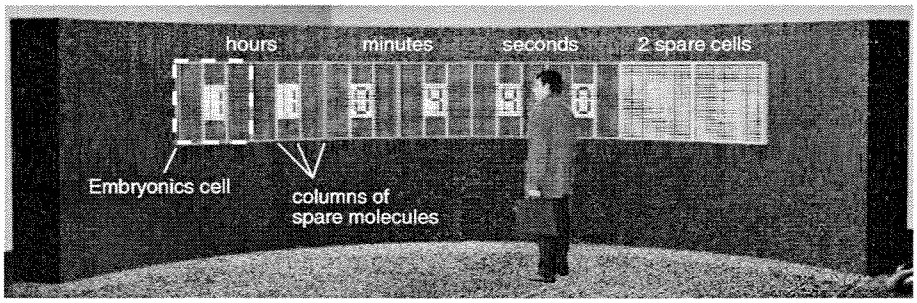

Figure 4.12: A computer graphic of the BioWatch.

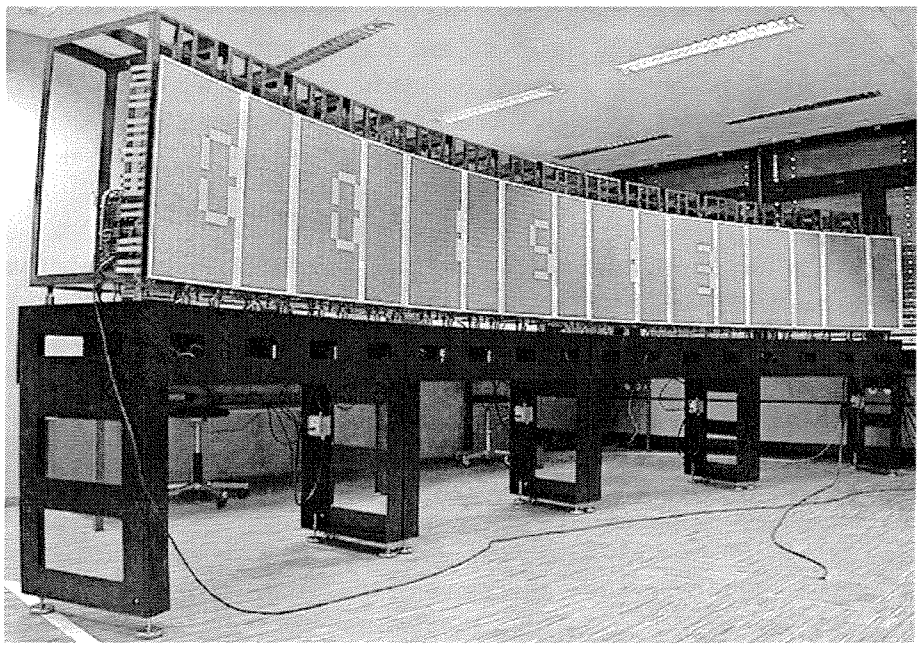

Figure 4.13: The BioWatch implemented on the real BioWall. Photo: André Badertscher.

A computer graphic of the BioWatch application is depicted in Figure 4.12, the real implementation is shown in Figure 4.13. Each of the six digits required to count the hours, minutes, and seconds represents an Embryonics

cell built up of about 400 Embryonics molecules, each molecule corresponding to the BioWall's fundamental element (Figure 4.5). In addition to the six cells, two spare cells are added to the right of the wall. Each unit of the BioWall is a molecule of the Embryonics hierarchy. A cell is then a mosaic of $20 \times 25 = 500$ molecules, and contains two repair columns ($2 \times 25 = 50$ molecules). The visitor has control over the "life" of each molecule. A fault can be inserted in any molecule simply by pressing on the corresponding unit's membrane. The fault detection mechanism included in the Embryonics molecular layer (embedded into the programmable logic of the Xilinx FPGAs) automatically detects the error and activates the molecular self-repair mechanism. A dead molecule is instantly replaced by the neighbor immediately to its right, and so on, until the nearest yellow repair column. The limits of this kind of self-repair imply that only a single molecule per line, between two repair columns, can be killed. If this constraint is respected, the cell survives any amount of faults, although the figure displayed is distorted. Each cell can thus tolerate up to two faults per line (one fault between each pair of yellow columns), i.e., $2 \times 25 = 50$ faults in total.

If the above rule is not respected, and several faults are inserted on the same line of the same cell between two repair columns, the molecules can no longer repair themselves and the cell dies. However, the death of a cell does not imply the death of the organism: it is instantly replaced by a spare cell to its right, while the dead cell is switched off and becomes a scar. It should be noted that, thorough this self-repair process, the BioWatch continues to work without fault: the tissue remembers its state and recovers the correct time after repair. Moreover, we are currently implementing an "unkill" mechanism that, should a sufficient number of faults be removed (by pressing the membrane of a dead molecule), will automatically re-activate a dead cell, which will recover its functionality (and its state) within the organism. The self-repair capabilities of the Embryonics machines are based on a general principle of life - cell differentiation. Each organism is a collection of cells, each containing a full copy of the genetic program, the genome. This structure makes the whole organism extremely robust, since each cell contains the complete plan and can therefore replace any other defective cell. Nevertheless, like all artificial and natural organisms, the death of a sufficiently large number of cells cannot be repaired, causing the death of the organism. Of course, the advantage of the "controlled" environment in which the machine operates is that the death of the organism causes a general reset of the system, the obliteration of all injected faults, and the "birth" of a new, perfectly functioning machine.

4.3.3 Turing Neural Networks

Recently, Turing's neural networks [408, 410] have been implemented on the BioWall's reconfigurable tissue (Figure 4.14). Each of the 3,200 units of the

machine can be interactively configured by choosing one out of five possible functions:

- empty cell,
- neuron,
- connection,
- synapse, or
- input cell.

The user (the external supervisor) is invited to discover and affect the behavior of the unorganized B-type machine by opening and closing synapses (i.e., "organizing" the machine) and by modifying the network's inputs. All modifications occur by simply pressing on the respective touch-sensitive membranes. This application is first and foremost a demonstration of Turing's neural networks on reconfigurable hardware (to the best of our knowledge, the first one). However, it also exemplifies the fusion of the ontogenetic and epigenetic models in a single artificial tissue.

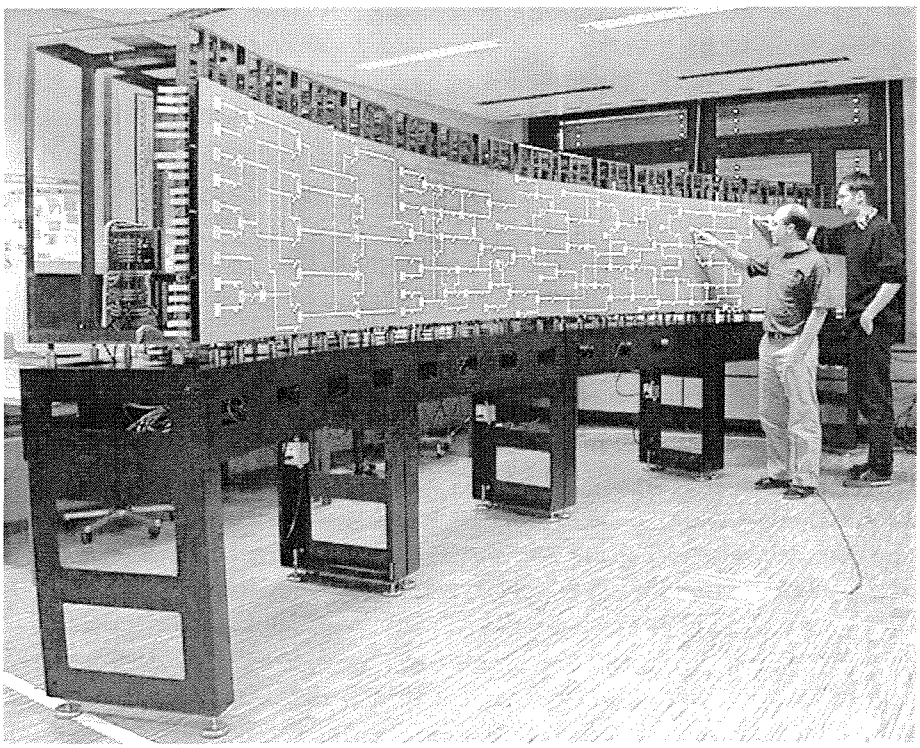

Figure 4.14: World first's implementation of Turing's neural networks. Photo: André Badertscher.

4.3.4 The Firefly Synchronization Task

In 1997, the Logic Systems Laboratory presented an evolving hardware system called *Firefly* [374] that successfully solved the synchronization task in one dimension using cellular programming based co-evolution. The novelty of Firefly was that it operates with no reference to an external device (such as a computer that carries out genetic operators) thereby exhibiting online autonomous evolution.

Recently, the synchronization task in two dimensions has successfully been evolved on the BioWall's 3200 FPGAs. The BioWall [404] is a giant reconfigurable computing tissue developed to implement machines according to the principles of the *Embryonics* (embryonic electronics) project [246]. The BioWall's size and features are designed for public exhibition, but at the same time it represents an invaluable research tool, particularly since its complete programmability and cellular structure are extremely well adapted to the implementation of many different kinds of bio-inspired systems. The implementation on the BioWall consists of a two-state, non-uniform CA, in which each cell (i.e., each FPGA of the BioWall) may contain a different rule. The cells rule tables are encoded as a bit-string, known as the genome, that has a length of $2^5 = 32$ bits for our 2D CA since the binary CA has a neighborhood of 5

To evolve the CA, Algorithm 4 of Section 3.5.3 has been used. Rather than to employ a population of evolving CAs, the algorithm evolves a single, non-uniform CA of the size of the entire BioWall (one cell of the CA in each unit of the BioWall, that is, 3200 cells), whose rules are initialized at random. For more details, see Section 3.5.3.

Using this cellular programming approach on the BioWall, we have shown that a non-uniform CA of radius 1 can be evolved to successfully solve the synchronization task. In addition, once a set of successful rules has been found, our machine allows the state of each CA cell to be changed by pressing on its membrane. The user can then observe how the machine synchronizes the 3200 cells.

In contrast to the original Firefly machine [374], which used a slightly simplified algorithm to facilitate the implementation, the BioWall implements exactly the algorithm as described above. The BioWall implementation, however, contains no global synchronization detector, therefore, it is not 100% guaranteed that the CA always synchronizes perfectly (i.e., for any initial configuration). Each cell contains a pseudo-random generator, realized by means of a linear feedback shift register, to initialize randomly the cell's state and its rule table. The register receives an initial random seed from an external PC that also configures the FPGAs. The actual implementation easily fits into a Xilinx Spartan XCS10XL FPGA.

Figure 4.15 shows a sequence of the synchronization task as implemented on the BioWall.

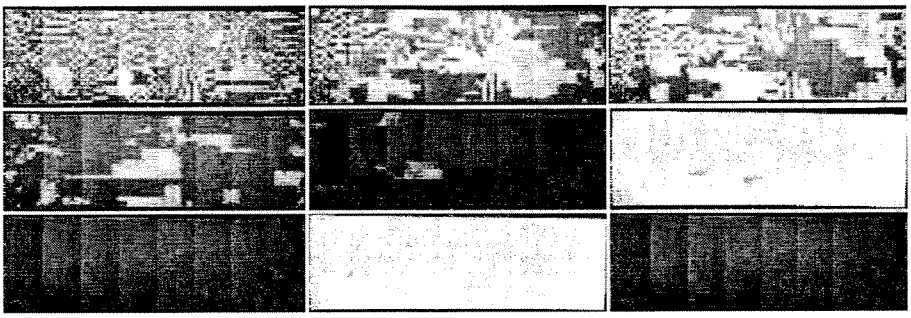

Figure 4.15: An implementation on the BioWall of the two-dimensional synchronization task. The images show a sequence of the synchronization. Photo: Christof Teuscher.

4.3.5 DNA Sequence Comparison

Nowadays the comparison and the alignment of characters taken from a finite alphabet is a fundamental task in many applications, ranging from full-text search to computational biology. In particular, string comparison is a critical issue in the field of molecular biology. In fact, both DNA fragments and proteins can be represented as sequences of characters (taken from alphabets of 4 and 20 symbols, respectively) and sequence similarities provide useful information about the functional, structural and evolutionary relationships between the corresponding molecules. Biological sequences may differ because of local substitutions, insertions and deletions of one or more characters. The complexity of string comparison comes from the large number of possible combinations of these three basic mutations.

The similarity between two strings can be evaluated either in terms of edit distance or in terms of similarity score. The edit distance is a measure of the minimum number of mutations that may have transformed the first string in the second, or, in other terms, of the minimum number of edit operations required to make the two strings equal to each other. The similarity score is a measure of the maximum number of residual matches between the two strings. Both metrics can be evaluated in polynomial time by means of dynamic programming techniques.

The key algorithm for evaluating the similarity between two strings of length N and M was developed by Needleman and Wunsch [286] and takes $O(N \times M)$ steps to complete execution. The two-dimensional structure of the algorithm (see [59]) makes it suitable for a parallel implementation on a systolic array. In particular, hardware parallelism can be exploited to perform string comparison in $O(N+M)$ steps. In this experience we present in [59] a parallel implementation of the Needleman-Wunsch algorithm on the BioWall (Figure 4.16).

The BioWall cannot compete in performance with existing parallel im-

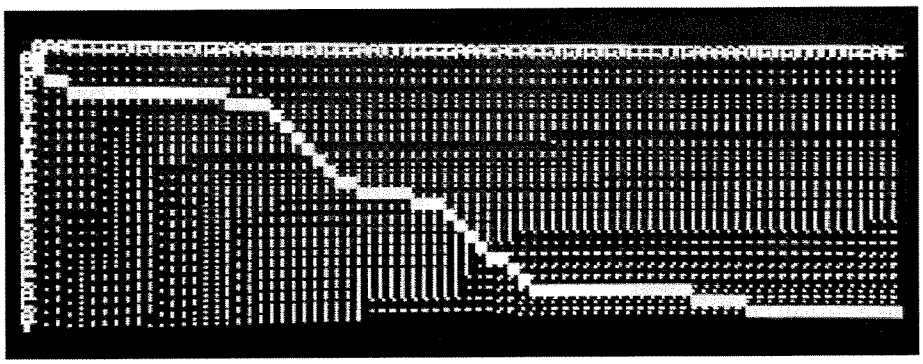

Figure 4.16: DNA sequence comparison on the BioWall. Photo: André Badertscher.

plementations of the Needleman-Wunsch algorithm, since it suffers from the typical performance limitations of a large prototyping platform. Nevertheless, the implementation of the Needleman-Wunsch algorithm on the BioWall is a significant design experiment in the field of reconfigurable computing because of the peculiarities of the target architecture.

4.3.6 Wrap-Up

The current BioWall is a mosaic of over three thousand transparent electronic modules. Each of them enables the visitor to communicate with the surface simply by touching it with their finger, calculates its new status and indicates it immediately on an electronic display. This extraordinary ability is demonstrated through a number of experiments.

In fact, the applications we presented are just a small sample of the capabilities of the BioWall, capabilities that we are still discovering. The cellular structure of the machine make it an ideal platform for the prototyping of bio- inspired systems, which often exploit this kind of structure, very common in nature at all levels. Its size and structure impose a certain number of limitations (e.g., clock speed), but its complete programmability (the entire surface is composed of Xilinx FPGAs) provides an outstanding versatility (the different applications we mentioned should be a sufficient, if incomplete, example) and the visual and interactive component of the system are invaluable tools both for the dissemination of ideas and for the verification of research concepts that are often limited to software simulations.

Among some the other bio-inspired applications that we have implemented or plan to implement on our machine we will mention, for example, L-systems, ant simulations, predator-prey environments, other kinds of CAs, and more "conventional" artificial neural networks.

4.4 The POEtic Tissue

The goal of the *POEtic* project[3] [405,428] is the development of a novel digital electronic circuit, a flexible computational substrate or artificial tissue, capable of integrating the three biological models of self-organization (see Section 2.7): phylogenesis (P), ontogenesis (O), and epigenesis (E). This tissue will be the essential substrate for the creation of POE-based machines, capable of evolution, growth, self-repair, self-replication, and learning. The POEtic tissue will be a cellular surface composed of a variable number of elements, or cells. Each cell will contain the entire description (genome) for the whole tissue and will have the ability to communicate with the environment (through sensors and actuators) and with neighboring cells and accordingly executing a function. During the project, the tissue will be validated on different kinds of applications that can benefit from its life-like properties, interacting with and adapting to their own environment.

The POEtic tissue will be a cellular surface composed of a variable number of elements, or cells. Each cell will have the ability to communicate with the environment (through sensors and actuators) and with neighboring cells (through bi-directional channels), and accordingly executing a function. Each cell of the tissue will have the same basic structure, but will be able to acquire different functionalities, as "totipotent cells" in living organisms.

This flexibility will be given by an organization in three layers (Figure 4.17):

1. a genotype plane,
2. a configuration plane, and
3. a phenotype plane.

The genotype plane of each cell will contain a full description of the organism in the form of a digital genome. The configuration plane will transform the genome into a configuration string directly controlling the processing unit of the phenotype plane. Through this cellular process, the tissue will be organized into a massively parallel multi-cellular electronic structure.

Within such structure, groups of cells will be able to co-operate to realize a given task, giving rise to substructures not unlike organs in living beings. It is important to notice that the tissue can implement the three models (P, O, and E) in any possible combination, providing a very powerful computational substrate.

The objectives of the POEtic project are as follows:

- To review evolutionary models applied to electronic circuits.

[3]For further information see: `http://www.poetictissue.org`

- To develop suitable genetic encodings and analyze their effects on fitness landscape.
- To develop a set of hardware mechanisms, inspired upon embryology, that will provide growth, self-repair, and self-replication of the electronic tissue.
- To review neuronal models and architectures suitable for digital implementation.
- To define and design O, PO, PE, OE, and POE cellular tissues including the properties defined earlier.
- To demonstrate the applicability of these tissues on a variety of applications where its effectiveness can be readily assessed qualitatively and quantitatively.

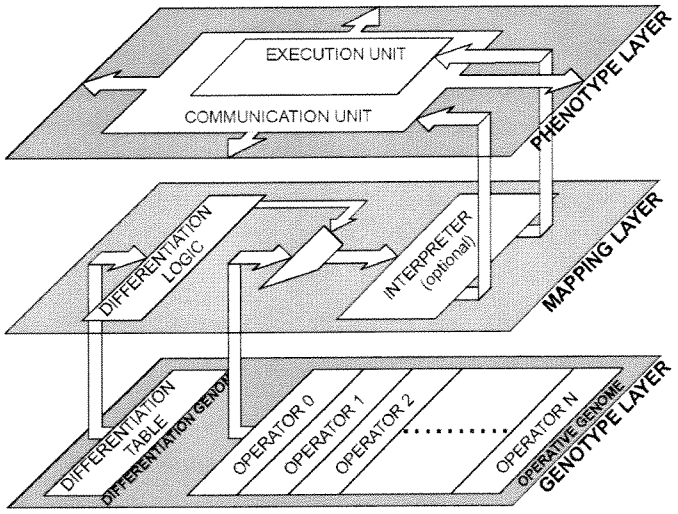

Figure 4.17: The layered organization of the POEtic chip.

4.5 Plastic Cell Architectures

The conventional computer (e.g., the von Neumann architecture) bears the advantage that any function can be realized by simply changing the computer's software. This is usually achieved through structure that separates the processing unit from the memory. The memory's contents can be changed and re-programmed ("plastic part") whereas the processing unit has fixed and pre-programmed ("built-in") functions.

The *Plastic Cell Architecture (PCA)*[4] [280,305] is a new general-purpose

[4]For further information see also: `http://www.onlab.ntt.co.jp/en/mn/pca/`.

computer architecture based on programmable logic. It consists of a dual-structured array of cells accommodating a fixed "built-in part" and a freely programmable "plastic part" (see Figure 4.18). The built-in part is based on a cellular automata model and serves as a communication structure. It is responsible for the configuration of the plastic part that consists of logic gates similar to programmable devices such as FPGAs. In contrast to the classical memory-processor von Neumann architecture, where the CPU operates as the main and central processing unit, the PCA uses the distributed logic circuits (i.e., the plastic parts) to compute a given task.

The problem with fine-grained distributed hardware is, however, always the same: programming the elements is highly nontrivial. The plastic cell architecture uses its own hardware description language called *Structured Function Description Language (SFL)* and a dedicated synthesis and CAD (computer aided design) tool called *PARTHENON*. SFL is an object-oriented register-transfer language that allows to describe the desired functionality at a high level of abstraction. PARTHENON takes an SFL description as an input and generates highly optimized circuits for various target technologies (ASIC, FPGA).

4.6 The Cell Matrix Architecture

According to the Cell Matrix corporation's website[5], its "[...] core technology is a convenient and elegant way to organize matter and energy to do computing. We have self-configurable hardware, and software that lets you configure it to implement circuits and systems. We also have developed software that takes full advantage of our hardware to do things that are painstakingly difficult to do today – faster, simpler, and cheaper."

The *Cell Matrix* architecture [117, 241, 242] (first also called *Processing Integrated Grid*) is an extremely fine-grained and universal hardware architecture not unlike an FPGA architecture. The reconfigurable device consists of a homogeneous collection of cells (Figure 4.19) that are interconnected in a nearest-neighbor scheme (von Neumann neighborhood; but other neighborhoods are easily realizable). Each cell performs very basic operations only that are based on a look-up table. Unlike traditional reconfigurable devices that are controlled by external systems, the Cell Matrix is self-configurable, i.e., each cell within the matrix can reconfigure other cells. In order to accomplish this, each cell can operate independently in one of two modes: (1) the D mode or (2) the C mode. In the D mode (processing mode), incoming data is processed by the cell's internal look-up table and used to generate the output. In the C mode (configuration mode), on the other hand, incoming data is used to re-write the cell's look-up table. A cell can therefore modify one of its neighbors by placing it in the C mode and by writing the data into

[5] `http://www.cellmatrix.com`.

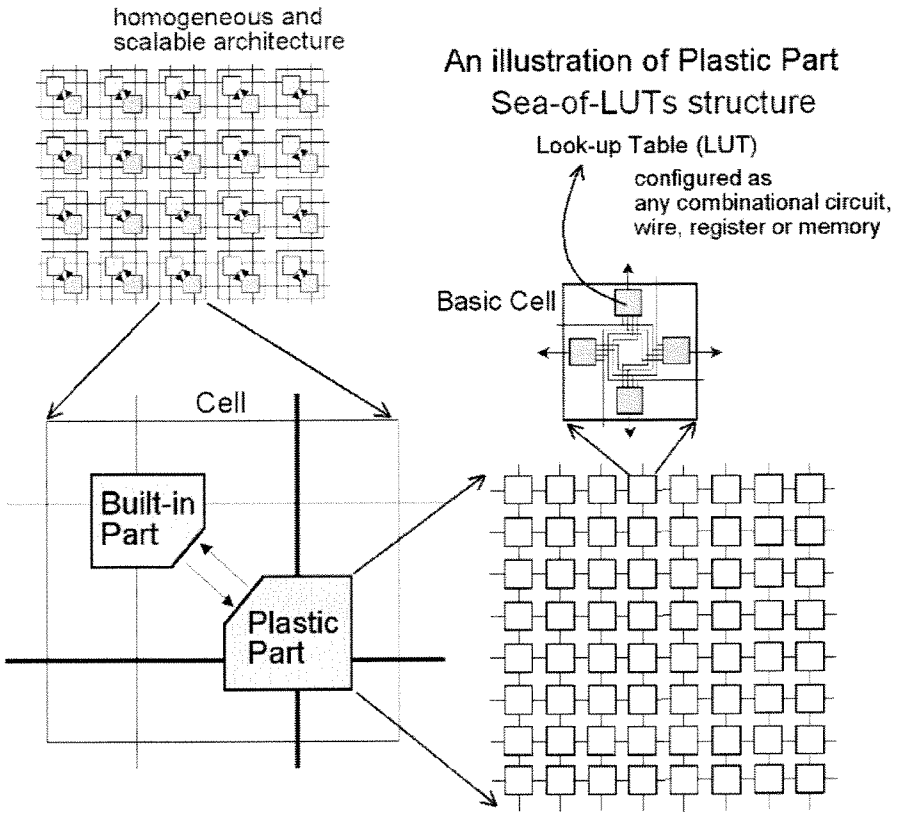

Figure 4.18: Structure of the *Plastic Cell Architecture (PCA)*. Reproduced from: `http://www.onlab.ntt.co.jp/en/mn/pca`.

the look-up table. As a cell's look-up table governs the ability to control a neighboring cell's mode, a cell X, for example, can configure a neighboring cell Y in such a way that Y will subsequently configure a third cell Z. By combining cells to so-called *Supercells*, hierarchical designs of almost any complexity can be realized. The small groups of cells then interact with nearby groups to achieve higher functionalities, which are used to perform more complex functions, and so on.

In [117], Durbeck and Macias hypothesize that the Cell Matrix architecture is an ideal architecture for atomic-scale technology such as nanotechnology, mainly because it is completely scalable (to an extreme number of transistors), self-reconfigurable, and configurable in parallel. The principal advantage of the architecture is that the hardware is (physically) simple and potentially very inexpensive to fabricate. In addition, a pretty powerful feature is that the configuration of its cells can be changed by the cells themselves through cooperation with neighboring cells. This however, requires very special design techniques and elaborates software tools.

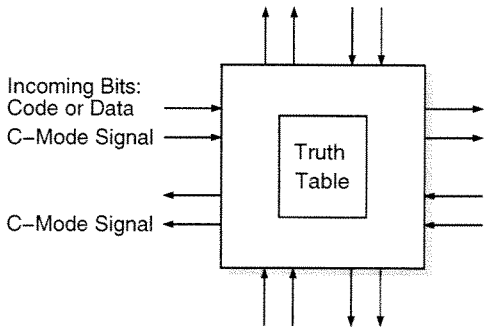

Figure 4.19: The basic cell of a Cell Matrix array.

The Cell Matrix's main drawback is that an enormous amount of cells is required to realize any practical system. This is mainly also due to the fact that cells do not contain any sequential logic, i.e., they only have the combinational look-up table. Therfore, even the realization of a D-flip-flop requires several cells. Finally, it is worth mentioning that the Cell Matrix functions very similar to the good old dataflow machine [439], which unfortunately never had a real breakthrough in computer architecture.

4.7 Programmable Matter

Programmable Matter[6] [417] refers to a very fine-grained computing medium that processes data arranged in a regular mesh or lattice in a way that is local, parallel, and physics-like. In a broader sense, programmable matter therefore refers to the computation of lattice-gas and cellular automata. Naturally, the computations such a system can perform are well suited for tasks that can be broken down as the composite of many, many simple, 'microscopic' computations. Indeed, gas, polymer, and magnetic spin system simulators as well as image processors, pattern generators, data encryptors and data compressors are all good examples of tasks a programmable matter can handle.

> In programmable matter, the same cubic meter of machinery can become a wind tunnel at one moment, a polymer soup at the next; it can model a sea of fermions, a genetic pool, or an epidemiology experiment at the flick of a console key" [417].

Programming matter, however, is nothing more — at least so far — that a cellular automata machine (CAM). The machine, described in [417] by Toffoli and Margolus, presents a hardware structure that is optimized for the

[6] `http://pm.bu.edu`

fine-grained simulation of spatially extended physical systems. The machine is essentially a fine-grained multiprocessor cellular automata machine.

Programming the matter is all but a trivial task. The program has to define the dynamics of a basic building-block (i.e., a cell) as well as the mesh interconnection topology. The input to the program is the initial configuration stored in the programmed matter mesh. The program runs for some number of time steps, during each of which all cells are being updated in parallel according to the dynamics. The output is the final state of the mesh.

4.8 A First Reconfigurable Membrane System Hardware Implementation for FPGAs

4.8.1 Introduction

The main goal of this section[7] is to present a hardware-friendly computational architecture for a certain class of membrane systems. The question of whether to simulate systems in software or to implement them in specialized hardware is not a new one (see for example [155]). With the advent of *Field Programmable Gate Arrays (FPGAs)* [421,441], however, this question took a back seat since it suddenly became easy and inexpensive to rapidly build (or rather *configure*) specialized hardware. Currently, P systems are usually implemented and simulated on a standard computer using an existing (such as the Iasi P systems simulator [70], or one of the recently proposed distributed software simulators [71,400]) or a custom simulator. As Paun states, "[i]t is important to underline the fact that "implementing" a membrane system on an existing electronic computer cannot be a real implementation, it is merely a simulation. As long as we do not have genuinely parallel hardware on which the parallelism [...] of membrane systems could be realized, what we obtain cannot be more than simulations, thus losing the main, good features of membrane systems" [315, p. 379]. To the best of our knowledge, a first possible hardware implementation has been mentioned in [243].

The hardware-based implementation presented in this section is a parallel implementation that allows to run a certain class of P systems in a highly efficient manner. The current design has been simulated and synthesized for FPGAs only, it could however very easily be used to implement an application specific integrated circuit (ASIC).

For this first implementation and in order to be able to efficiently implement P systems in hardware, we had to modify "classical" P systems (see

[7] I am indebted to Petreska Biljana for the implementation. Many thanks! A first draft of this section has been presented at the MolCoNet workshop on membrane computing (WMC2003), Tarragona, Spain.

Section 2.11) in the following two points:

1. The rules are *not* applied in a maximum parallel manner but following a predefined order.
2. The P system is *deterministic*, i.e., for a given initial configuration, the systems always halts in the same halting configuration.

The main reason is that a straightforward implementation of classical P systems would have been too expensive in terms of hardware resources needed. We were basically interested by a minimal hardware architecture and not primarily by a faithful classical P systems implementation. More details on the reactor algorithm will be given in Section 4.8.6.

4.8.2 Description of the Implementation

In this section, the hardware implementation of the membrane system shall be described in detail. The implementation basically supports cooperative P systems with priority using membrane dissolution and creation. Note that the source code of the implementation and further information is freely available on the following web-site: `http://www.teuscher.ch/psystems`. The resulting design is a universal membrane module that can be instantiated and used anywhere in a membrane system.

For the current implementation, we have chosen the FPGA technology as it is an ideal platform for prototyping hardware systems, is fully reprogrammable, and offers very good performance at a low cost. A *Field Programmable Gate Array* (FPGA) circuit [353,421,447] is an array of (a usually large number of) logic cells placed in a highly configurable infrastructure of connections. Each logic cell can be programmed to realize a certain function (see also [421] for more details).

In addition, once a design has been implemented on an FPGA, it can rather easily be transferred to a full custom *Application Specific Integrated Circuit* (ASIC) technology, which usually provides even better performance.

4.8.3 Membrane Structure

The membranes of our implementation are without a material consistence, i.e., they do not exist as physical frontiers. Therefore, when referring to the membrane, we will actually refer to its *contents*, i.e., to the multisets of objects and the evolution rules of the region it encloses (see Figure 4.20). The membrane's main functionality of physically separating two regions can be completely and unambiguously replaced by representing the relations between the membrane's contents (see Figure 4.21, A). According to Paun's membrane definition, the size and the spatial placement (distances, positions with respect to any coordinates) do not matter [315, p. 51] and we

therefore decided to dispose the membranes in the two-dimensional space in an arbitrary manner. The only information that matters and that allows to preserve the membrane structure are the relations (i.e, the interconnections and the hierarchical organization) between the cellular membranes (see Figure 4.21, B).

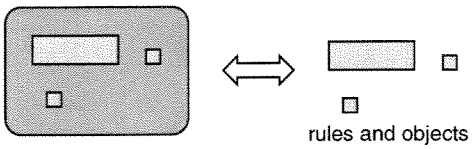

Figure 4.20: Membranes are not explicitely represented, instead, the cell is defined by its contents.

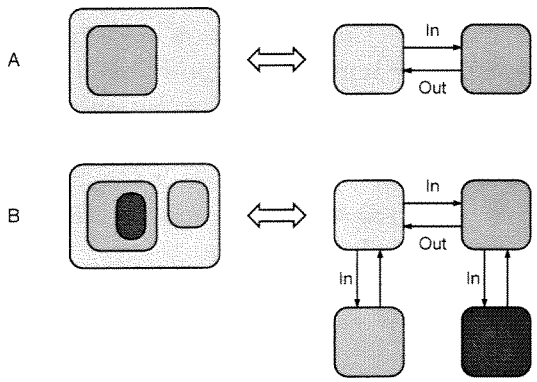

Figure 4.21: The relations between the membranes are represented by the interconnections.

The relations between the membranes—which are represented in our implementation by electronic buses (a collection of wires)—are basically only used when objects are being transferred between two membranes. A membrane can send and receive objects only from its external enclosing membrane and from the membranes it contains on the next lower level. For example: membrane 4 of Figure 2.27 can send and receive objects from its external membrane 1 (upper-immediate) and from 5, 6, and 7 (lower-immediate). However, membrane 4 cannot directly communicate with membranes 8 and 9 nor with 2 and 3. Therefore, connections in the form of bi-directional communication channels must exist between a membrane, its lower-immediate and upper-immediate membranes (see Figure 4.23, A). In the following, the two communication directions shall be described.

First, one communication channel (i.e., an electronic bus) is required to *send* objects to the lower-immediate membranes (see Figure 4.23, B). The bus originates from the upper membrane (1) and all the lower membranes are

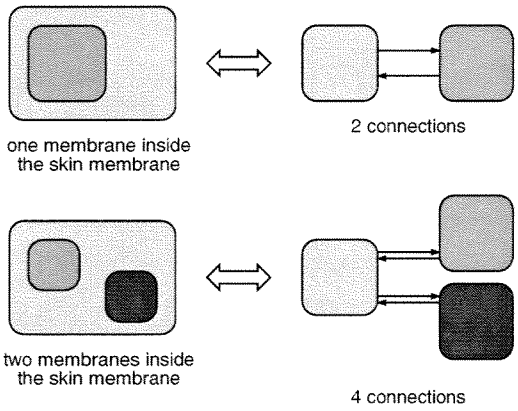

Figure 4.22: The number of connections basically depends on the membrane structure, which represents a serious problem for hardware implementations. Our implementations avoids this problem by using a special bus.

connected to it. It is possible that several lower-immediate membranes are connected to the same bus (e.g., membranes 2 and 3). To directly address a given message to a specified membrane, a label is attached to each message that is sent on the bus. Each membrane compares the label of each message with its own label and accordingly decides whether it is the receiver or not.

Second, to *receive* objects from the lower-immediate membranes, one bus for each lower-immediate membrane would basically be required as objects can be received simultaneously (Figure 4.23, C). We did not adopt this solution as it is dependent on the membrane structure. To avoid multiple buses, we simply replaced them by a bus that first "traverses" all the lower-immediate membranes (i.e., membranes 2 and 3) before it gets connected to the target membrane (1). Each lower-immediate membrane basically adds its objects to the message that is transferred on the bus before passing it on to the next membrane in the chain.

Naturally, a membrane can be at the same time an upper-immediate (if it contains other membranes) and a lower-immediate for another membrane (membrane 2 of Figure 4.23, D, for example). Therefore, each membrane must basically possess the connections required to connect to both of these levels (Figure 4.23, E and F), although some of them might not always be used.

Our architecture is completely independent of the surrounding membrane structure. For example, as shown in Figure 4.22, the number of interconnections is basically dependent on the number of the lower-immediate membranes. Each time a new membrane is being created, a new bus would have been required. Our final membrane implementation—as shown in Figure 4.23, F—is a sort of universal component which avoids this serious drawback and which might therfore be used anywhere in the hierarchy of a mem-

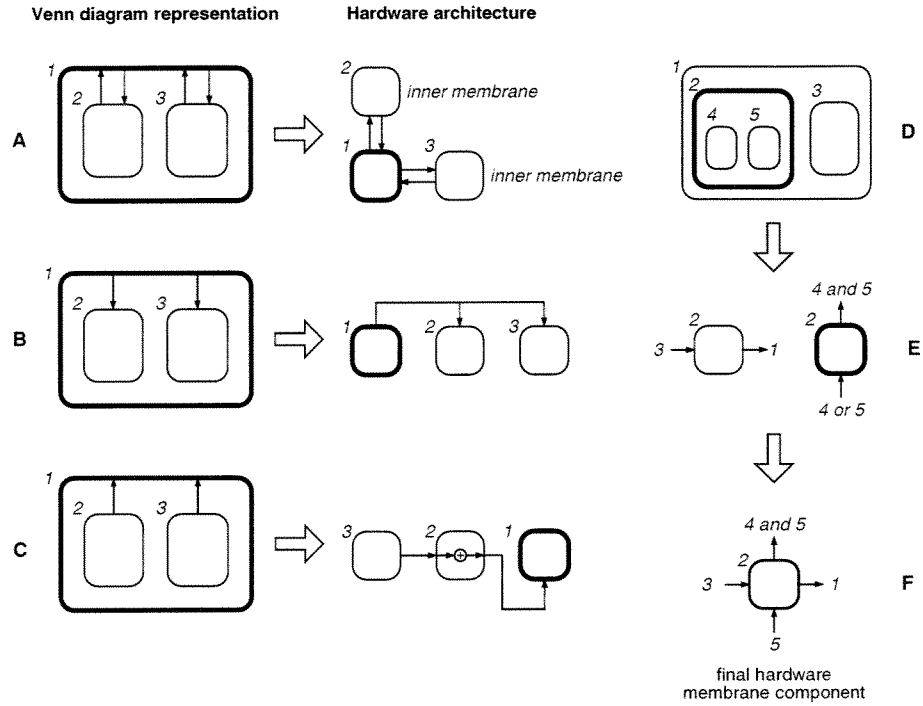

Figure 4.23: Possible connection structures between membranes. Sub-figure F shows the final membrane hardware component with its interconnection structure. The membrane component is universal and might be used anywhere in the hierarchy of a membrane system.

brane system.

4.8.4 Multisets of Symbol-Objects

Each region of a membrane can potentially host an unlimited number of *objects*, represented by the symbols from a given alphabet. In our case, these objects are not implemented individually but only their *multiset* is represented. The multiplicity of each set, i.e., number of identical objects, is stored within a register[8] as shown in Figure 4.24. The register's order reflects the order of the objects within the alphabet and consequently, the register position directly indicates which symbol's multiplicity is being stored. Note that the size of the register limits the maximum number of object instances that can be stored. In our implementation, you can choose between 8-bit and 16-bit registers, which allows to store up to $2^7 = 128$ or up to $2^{15} = 32768$ instances of an object (the remaining bit is being used to detect a capacity overflow).

[8] A memory element with rapid access.

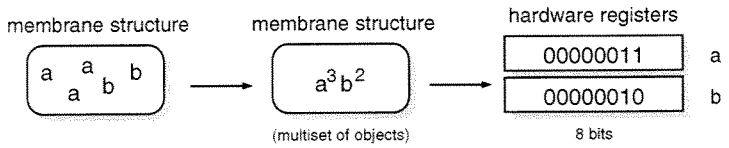

Figure 4.24: Representation of the symbol-objects in hardware. Alphabet = {a, b}.

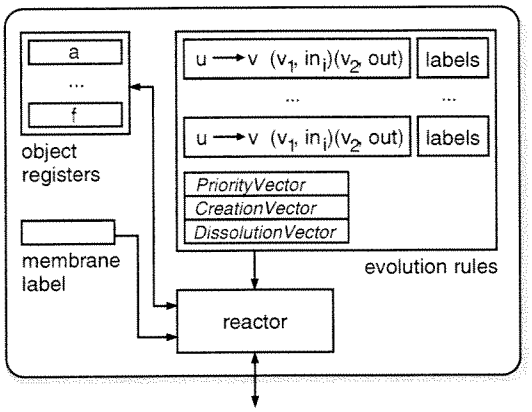

Figure 4.25: A view within the membrane hardware component.

4.8.5 Evolution Rules

An evolution rule is a rewriting rule of the form $u \rightarrow v(v_1, in_i)(v_2, out)$ that replaces the membrane objects u by the objects v, sends the objects v_1 to the lower-immediate membrane with label i, and sends the objects v_2 to the upper-immediate membrane. *Cooperative* rules, i.e., rules with a length of u greater than 1, are also supported in our implementation. There is only one possible *in target* command in our implementation, whereas in some P systems there may be several. However this simplification does not significantly affect the computational power and could easily be modified in a future extension of the design.

The module responsible for the evolution rules (see also Figure 4.25) is composed of the following elements (for more details on the priority and the membrane dissolution and creation, see also Sections 4.8.7 and 4.8.8):

- 4 arrays of 8-bit registers (4 multisets of objects) store the *left-hand side*, *here*, *out* and *in* parts of a rule (i.e., u, v, v_1, and v_2 in the formula);
- one 8-bit register contains the *in target* label (i.e., i from in_i);
- 8-bit registers store up to 4 labels of rules with higher priority;

- one 8-bit register specifies the label of the membrane to be created (in case of a creation rule);
- one bit in the 8-bit *PriorityVector* register is set to "1" if the rule is priority-dependant;
- one bit in the 8-bit *DissolutionVector* register indicates whether the rule will dissolve the membrane; and
- one bit in the 8-bit *CreationVector* register indicates whether the rule will create a membrane.

All these registers are initialized only at the beginning of the simulation and do not change afterwards. Remember also that the P system's evolution rules do not change throughout a computation, except for dissolving membranes, where the associated set of rules disappears.

In addition, a further module constantly computes whether a rule can be applied or not. The module takes the membrane objects as inputs and generates the signal *Applicable* for each rule:

$$Applicable\,(u \rightarrow v) = 1 \qquad \text{iff} \qquad u \leq w \tag{4.1}$$

where w is the multiplicity of membrane objects and $u > 0$. For example, the rule $a^3 \rightarrow b$ can only be applied if there are at least 3 objects of a present within the current membrane. Connecting all the *Applicable* signals of all the rules present in the membrane together by means of a logical or-gate then directly provides the signal indicating when the membrane's computation is finished, i.e., when all rules (that can be applied) have been applied.

4.8.6 Reactor Algorithm

The evolution of a P system can basically be decomposed into *micro-* and *macro-steps*. We used a somehow different decomposition as the micro- and macro-steps proposed by Paun in [314]. Our micro-step corresponds to the application of a rule inside a membrane. A macro-step then consists in applying sequential micro-steps in parallel to all membranes, as long as a rule can be applied somewhere. At the end of a macro-step (i.e., no further rule is applicable), the newly obtained and the stored objects are consolidated to the already existing objects (in all membranes in parallel). The system knows when all micro-steps have been completed due to the global signal generated on the basis of the membrane's *Applicable* signals (see Section 4.8.5). Algorithms 7 and 8 illustrate the details of the micro- and macro-step in our hardware implementation. Note that the rule r is of the form $u \rightarrow v(v_1, in_i)(v_2, out)$ and the "store" operation simply updates the internal registers.

Algorithm 7 Micro-step of our P system

Select a rule r
if *Applicable*(r) **then**
  Remove left-hand side objects u from membrane objects
  Store right-hand side objects v in the *UpdateBuffer*
  Send v_1 to the lower-immediate membranes (via the connection bus)
  Store v_2 in the *ToUpperBuffer*
end if
Store objects from upper-immediate membrane in the *FromUpperBuffer*

Algorithm 8 Macro-step of our P system

In all membranes simultaneously
while Exists an applicable rule in the system **do**
  Perform a micro-step
end while
Send *ToUpperBuffer* objects to the upper-immediate membrane
Add *UpdateBuffer* objects to membrane objects
Add *FromUpperBuffer* objects to membrane objects
Add objects from the lower-immediate membrane to membrane objects

The selection of the rules (first step of Algorithm 7) is done in the following way: a special memory inside the membrane contains an ordered list of rule labels (specified at the P system initialization). This sequence might either be generated randomly—which simulates in a certain sense a nondeterministic behavior—or in a pre-specified order. However, it is important to note that two equal initializations result in the same P system behavior, i.e., the system is completely deterministic, which is different from the original P systems idea. Furthermore, the rules of a P system are usually applied in a maximum parallel manner. As Algorithm 7 illustrates, our P system hardware implementation does not allow a maximum parallel application of the rules within a micro-step. The main reason for this was that such an algorithm would have been too expensive to implement in terms of resources used as it would require to implement a search algorithm. However, the parallelism on the membrane level is fully realized and membranes compute simultaneously.

4.8.7 Priorities

A rule $r_1 : u_1 \rightarrow v_1$ from a set R_i is used in a transition step with respect to Π (the membrane system) if there is no rule $r_2 : u_2 \rightarrow v_2$ which can be applied at the same step and $r_2 > r_1$ [315, p. 70] (i.e., r_2 has a higher priority than r_1). Remember that there is competition for the rule application and

not for the objects to be used. In order to respect the priorities, the priority-applicability of a rule is computed only once, i.e., at the beginning of each macro-step. The signal *PriorityApplicable* is generated according to the following formula:

$$PriorityApplicable\,(r_\mathrm{i}) = \neg\left(\sum_\mathrm{j} Applicable\,(r_\mathrm{j}) \mid r_\mathrm{j} > r_\mathrm{i}\right) \tag{4.2}$$

The information $r_j > r_i$ is contained in the *labels* (see Section 4.8.5 and Figure 4.25). Once selected, a rule is applied only if it is both *Applicable* and *PriorityApplicable* (i.e., both signals are active).

4.8.8 Creating and Dissolving Membranes

The work presented so far did not allow to create and dissolve membranes. We therefore propose these two extensions to our membrane hardware architecture. Dissolving and creating membranes is both biologically and mathematically motivated. As Paun writes, it is a frequent phenomenon in nature that a membrane is broken, and that the reaction conditions in the region change and become those of the covering membrane, for all the contents of the previously dissolved membrane. Furthermore, membranes are also created continuously in biology, for instance in the process of vesicle mediated transport. When a compartment becomes too large, it often happens that new membranes appear inside it. From an organizational point of view, membrane creation is mainly used as a tool for better structuring the objects of a given (natural or artificial) system. But interestingly, we observe also an increase of the computational power since the class of creating membranes can solve NP-complete problems in linear time (see [315, p. 311] for an example of linearly solving the SAT problem by means of P systems with membrane creation).

The first extension is the *membrane dissolving action* [315, p. 64]. The application of rules of the form $u \to v\delta$ in a region i is equivalent to using the rule, then dissolving the membrane i. All objects and membranes previously present in the membrane become contents of the upper-immediate membrane, while the rules of the dissolved membrane are removed.

The solution adopted for our hardware implementation simply consists in disabling the membrane by means of a membrane *Enable* signal. Once a membrane dissolving rule has been used, the membrane objects are sent to the upper-immediate membrane, the membrane is reconfigured to fit the new connection configuration, and the enable signal is set to "0" to "disable" the membrane. It is important to note that the membrane (i.e., its contents) continues to physically exist. This is, among other reasons, necessary to in order not to interrupt the communication busses as presented in Section

4.8.3. Obviously, the drawback of this solution is that the space occupied by a "removed" membrane cannot be reused.

The second extension made to our model consists in the creation of membranes [315, p. 302]. When a rule of the form $a \rightarrow [_i v]_i$ is used in a region j, then the object a is replaced with a new membrane with the label i and the contents as indicated by v. The label i is important as it indicates the set of rules that applies in the new region.

The solution adopted for our hardware implementation consists in creating the potentially "new" membranes at initialization already. This is possible since all parameters needed for their creation (i.e., their objects, evolution rules, and the position in the membrane structure) are already known. At the beginning, these membranes are considered as non-operational. Once a membrane creating rule has been used, one of the already created membranes will simply be activated (until they are all used). Obviously, the number of available inactive spare membranes limits the computational power of the system.

4.8.9 Design Flow and Java-Tool

The membrane system has been programmed using the VHDL [17,447] hardware description language. The FPGA programming process basically requires three software tools (note that many other tools exist): ModelSim[9] (a VHDL simulator), LeonardoSpectrum[10] (a VHDL synthesizer), and the Xilinx Design tools[11]. Figure 4.26 illustrates the typical design flow.

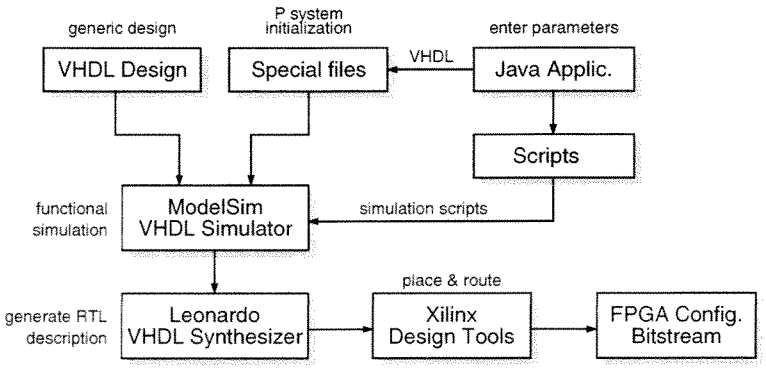

Figure 4.26: Illustration of the typical design flow.

The VHDL files are first compiled with the ModelSim software, which allows to simulate and debug the VHDL code on a behavioral level. Once compiled and simulated, the design is synthesized, analyzed, and optimized

[9]Model Technology: `http://www.model.com`.
[10]Mentor Graphics: `http://www.mentor.com/leonardospectrum`.
[11]Xilinx: `http://www.xlinix.com`.

by means of the LeonardoSpectrum software. LeonardoSpectrum also provides all technology-relevant details with regards to the chosen hardware technology or circuit, i.e., the electrical schemas, the timing analysis, the hardware resources required, etc. In addition, LeonardoSpectrum also outputs the EDIF file which is used in the next step by the Xilinx software. The Xilinx Design tools basically map (i.e., place & route) the design on the chosen chip and generate the necessary configuration files. In our case, the software outputs the configuration bitstream for the Xilinx FPGA we have chosen as a target technology.

Since a large part of the code is dependent on the membrane system to be simulated and on its initial values, we implemented a Java application which basically automates the generation of the VHDL source code. As shown in Figure 4.27, the application allows to specify in a convenient way all relevant parameters of the membrane system and then automatically generates the VHDL configuration and initialization files as well as several scripts that automate the simulation process.

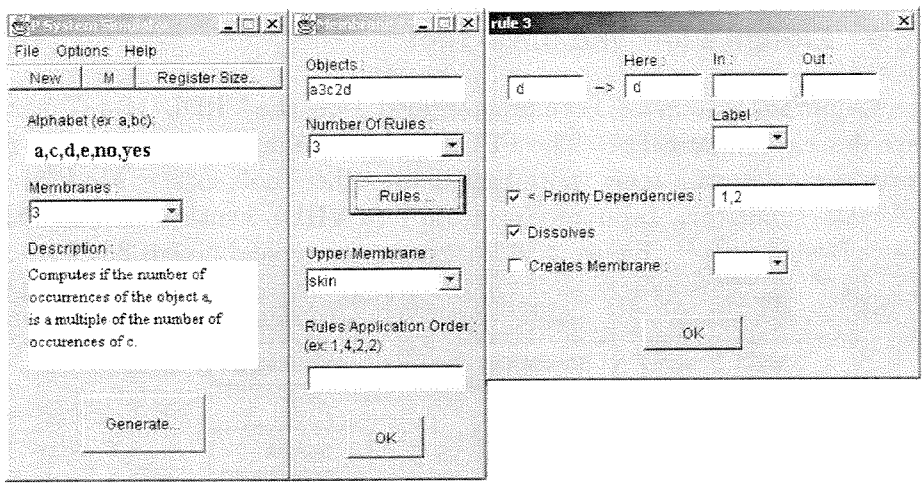

Figure 4.27: Java application that allows to automatically generate all necessary configuration files of the membrane system hardware implementation.

4.8.10 Experiments and Results

In this section, a simple educative toy example as well as a selection of results shall be presented.

An Example

We consider the P system presented on page 73 in [315] (see also Figure 4.28) that allows to decide whether k divides n. Figure 4.29 shows parts of the

timing diagram of a ModelSim simulation of this P system. The alphabet used is the following: $\mathsf{Alphabet} = \{\mathsf{a}, \mathsf{c}, \mathsf{c}', \mathsf{d}, \mathsf{no}, \mathsf{yes}\}$. The signal names as shown in the simulation represent the following information:

- **clk**: Global clock signal.
- **macro_step**: Set to '1' when a macro-step is being completed and then triggers the consolidation of the objects in all membranes.
- **values i**, and **newvalues i**: Correspond to the multiplicities of objects in the membrane i. *Values* i represents the objects currently contained in membrane i. *Newvalues* i represents the objects that will appear in membrane i after a rule (in this or another membrane) has been applied.

Note that the rules are not represented in Figure 4.29 as they do not change during the P system evolution. A description of the simulation—which requires a total of 15 clocks steps to complete—is as follows:

1. The system is initialized. The inner membrane with label 2 contains a^3c^2d (*values2*, `init`). The system therefore decides whether $n = 3$ divides $k = 2$.
2. Rule $ac \rightarrow c'$ is applied. The objects a and c in the second membrane are replaced by one object c', i.e., *values2* has been decremented on the positions that correspond to the objects a and c (follow the alphabet order) and *newvalues2* has been incremented at the position of the object d. Note that the rule $d \rightarrow d\delta$ cannot be applied in this macro-step as a rule of higher priority ($ac \rightarrow c'$) is applicable.
3. No rules can be applied in any membrane, therefore a new macro-step is started and the *values* are updated: *values* := *values* + *newvalues*.
4. The rule $ac' \rightarrow c$ is applied (the only applicable rule). Again, a macro-step and an update as in the previous step follow.
5. The rule $d \rightarrow d\delta$ is finally used as all the rules of higher priority cannot be applied (i.e., there are no more occurrences of the object a). This causes membrane 2 to dissolve in the next macro-step.
6. Beginning of a macro-step. Membrane 2 is dissolved, therefore its contents are transferred to membrane 1 (since membrane 1 contains membrane 2).
7. The rule $dcc' \rightarrow (no, in3)$ is applied in membrane 1 for it has priority over the other rule. The objects dcc' in membrane 1 are replaced with *no* (`result`) in membrane 3, which is the output membrane.

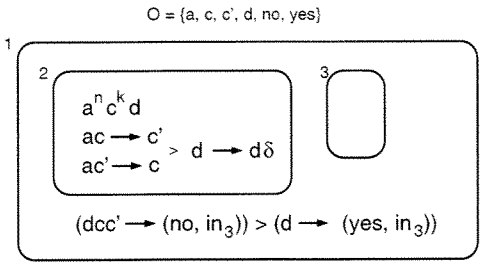

Figure 4.28: The membrane system simulated (page 73 in [315]).

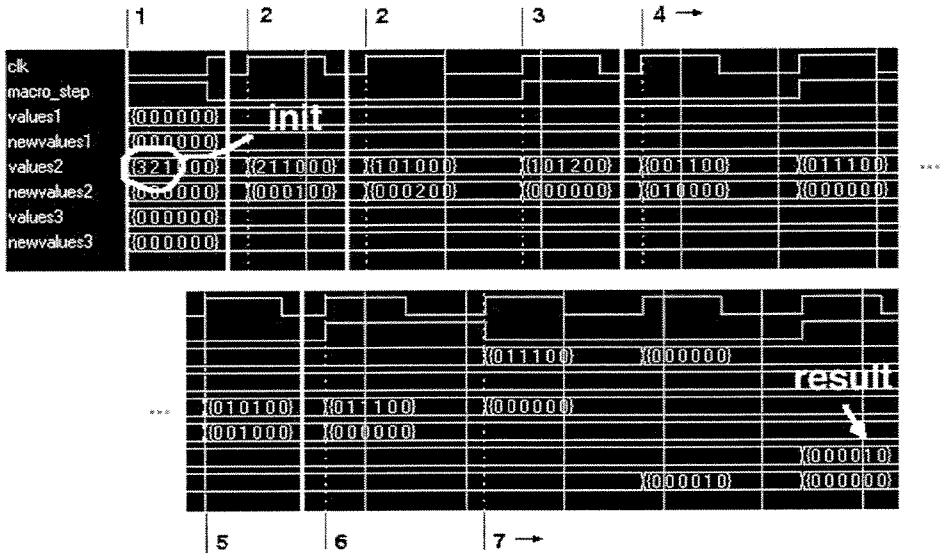

Figure 4.29: Screenshot of a ModelSim P system simulation. The white fat vertical lines indicate cuts on the time axis, i.e., irrelevant parts of the simulation have been removed.

Our implementation has been tested by means of four examples from [315] with the following features: membrane dissolution, membrane creation, transferring objects to upper- and lower-immediate membranes, and cooperative P systems with priorities. Short descriptions of the problems tested, screenshots of the ModelSim simulations, and the simulation results of the LeonardoSpectrum synthesis can be found on: `http://www.teuscher.ch/psystems`.

Results

In order to evaluate the performance of our hardware implementation, we synthesized membrane systems of different sizes and complexities. The tar-

get circuit was a Xilinx high-end Virtex-II Pro 2VP50 FPGA[12] that roughly contains about 24000 configurable logic blocks (CLBs)[13]. Table 4.1 summarizes the results obtained. One can see that the resources used are directly proportional to the maximum number of objects. Furthermore, adding the possibility of membrane creation adds complexity and therefore results in a design that is almost twice as large and runs at a much slower clock speed.

At the exception of the limited physical resources of the reconfigurable circuit, i.e., the number of gates available, there is no limitation on the number of membranes, rules, and objects that might be simulated from a theoretical point of view since the universal membrane architecture is entirely parameterizable by generic parameters in the VHDL code.

4.8.11 Wrap-Up

A universal and massively parallel hardware implementation of a special class of P systems has been presented. The proposed hardware design is highly parameterizable and can in principle be used to evolve membrane systems of any complexity as long as the underlying hardware provides sufficient resources. The architecture of the universal membrane hardware module allows to use the same module anywhere in a membrane system and independently of the number of rules and objects to be stored within it. The results have shown that membrane systems can be implemented very efficiently in hardware.

The drawback of the proposed implementation is its limitation to a special class of membrane systems.

Future work might on one hand concentrate on the development and improvement (in terms of speed and resources used) of the existing design, on the other hand, it would also be necessary to find useful (killer-) applications that require large and complex membrane systems that would fully exploit the specialized hardware.

4.9 Wrap-Up: Hardware or not?

Hardware implementations play a crucial role (in research) to accelerate, validate, and evaluate the performance of conceptual architectures. This chapter provided a certainly incomplete overview on some of the most relevant, alternative, and partially biologically-inspired hardware architectures.

My personal experience — and actually it is rather a generally valid matter of fact — with hardware is, however, also characterized by what AI guru Marvin Minsky brings to the point: "The worst fad has been these stupid little robots. Graduate students are wasting 3 years of their lives

[12] `http://www.xilinx.com`.

[13] 1 CLB = 4 logic cells, 1 logic cell = 4-input LUT + FF + Carry Logic.

soldering and repairing robots, instead of making them smart. It's really shocking".[14] This might of course not only be said for robots but also for any other real hardware realization. This is not to say that it's not worth doing any implementations at all, but one has to be very careful when deciding and evaluating the goals, advantages, and the resources required for such an "adventure". It is not at all uncommon that, when a hardware project is finally finished, it is completely outdated already.

Unfortunately, most alternative hardware architectures do in general not attract much attention in the scientific community, despite their originality. An important — and only too often neglected — issue is certainly the software. Most alternative hardware architectures require highly specialized and complicated software tools to use/program/configure the hardware. The absence of such tools does of course not help to gain attention in the research community or in industry.

Many an architecture is motivated by the hypothesis that future atomic-scale technologies would require new engineering methods — and I basically fully agree on this —, for example to build perfect systems out of billions of partially faulty elements. It is, however, in most cases not quite clear how the proposed architecture could be transferred to nanotechnology or any other new substrate different from silicon. One has to be clearly aware that the nano-world *is* very different from the silicon-world. As Stan *et al.* [384] state, for example, randomness is a matter of fact in the field of nanotechnology with which one has to deal with! In addition, most approaches are likely to fail as they do not offer full scalability and will be limited by the software tools required to program the myriads of elements (at least the architectures that do not offer parallel self-(re)configuration and/or replication).

[14] `http://www.wired.com/news/technology/0,1282,58714,00.html`.

Membranes	Objects	Resources (CLBs)	Clock frequency	Remark
10	6	1037 (4.2%)	198 MHz	dissolution only
10	12	2213 (9%)	190 MHz	dissolution only
10	6	2069 (8.4%)	45 MHz	dissolution and creation
10	12	4254 (17.3%)	31 MHz	dissolution and creation
20	6	1977 (8%)	198 MHz	dissolution only
20	12	4185 (17%)	191 MHz	dissolution only
20	6	3967 (16.1%)	45 MHz	dissolution and creation
20	12	8110 (33%)	27 MHz	dissolution and creation

Table 4.1: Summary of the performance results obtained for a Xilinx Virtex-II Pro 2VP50ff1517 (Speed grade: -7) FPGA.

CHAPTER 5

From Blending to Membrane Blending

There are no words in nature.

Hamish Fulton

5.1 Introduction

Cognitive science is an interdisciplinary field that has arisen during the past decade at the intersection of a number of existing disciplines: psychology, linguistics, computer science, philosophy, physiology, and neuroscience. The interest of cognitive science is understanding the nature of the mind, which is an old quest, dating back to antiquity in the case of philosophy. Previously, each discipline sought to understand the mind from its own perspective, benefiting little from progress in other fields because of different methods employed. With the advent of cognitive science, however, common interests and theoretical ideas have overcome methodological differences, and interdisciplinary interaction has become the hallmark of this field.

It was in the 1940s and 1950s that important developments paved the way for today's cognitive science. The most significant event is probably the conceptual invention of modern computing machines by Alan Turing [409] and their physical realization shortly thereafter. With the advent of the modern computer, a new field dubbed "artificial intelligence" also began to progress.

The closely related idea that mental activity could be described as information processing emerged in psychology at about the same time and is

still a highly disputed field. However, recent advances in brain scanning and imaging, such as computer-assisted tomography (CT), magnetic resonance (MR), and positron emission topography (PET) methods, have allowed human brains to be studied in ways heretofore impossible. For example, scientists can now identify specific regions of brain damage in neurological patients so that symptoms can be correlated with anatomical location. Using these methods in conjunction with those of cognitive psychology, cognitive neuroscientists are beginning to map out the function of major areas of the human brain.

It was mostly during the 1970s that researchers started to recognize the relevance of work in neighboring disciplines and to learn something about it, which marked somehow the true birth of cognitive science as a scientific endeavor.

In this chapter, an attempt to apply a theory developed in cognitive science (blending) as an organizational principle in a purely computational environment (membrane systems) is presented. After the presentation of semantic networks and the Copycat project, conceptual integration (or blending) is presented in detail. In the second part of this chapter, the computational blending approach entitled C-Blending is introduced, which makes use of analogies between membrane systems, artificial chemistries, and mental spaces.

5.2 Two Examples of Related Cognitive Architectures

I have chosen two examples of related cognitive architectures as an introduction to the topic: namely semantic networks and the Copycat project. The interested reader is also referred to `http://ai.eecs.umich.edu/cogarch0/index.html` for a survey of cognitive and agent architectures.

5.2.1 Semantic Networks

In 1968, Ross Quillian [329] introduced *semantic networks* as a method of modeling the structure and storage of human knowledge in the shape of a graph. Quillian wanted his system to explore the meaning of English words by the relationships between them. In particular Quillian's system sought to compare words and express the results of those comparisons. A semantic network is basically a tool for logical reasoning that concentrates on categories of objects and the relations between them. Computer implementations of semantic networks were first developed for artificial intelligence and machine translation, but earlier versions have also been used in other domains such as

psychology and linguistics. The semantics of a simple semantic network language can be stated by providing first-order logical equivalents for assertions in the network language [350]. What is common to all semantic networks—many different kinds exist—is a declarative graphic representation that that can be used either to represent knowledge or to support automated reasoning. An example is provided in Figure 5.1.

The semantic network architecture originally proposed by Quillian is composed of the following elements:

- *Nodes* represent a *concept* or word that the system "knows" about.
- *Edges* between nodes represent a *relation* between two concepts. Quillian originally had six categories of edges:

  1. Superclass/Subclass (the ISA relationship)
  2. Modification (an adjective or adverb)
  3. Disjunctive (logical OR; earth, air, fire, water)
  4. Conjunctive (logical AND; old red house)
  5. Unspecified binary predicate
  6. Unspecified binary predicate

  Note that the 5 and 6 are open ended, linking two thing concepts to a relationship concept to form a sort of custom link.

- A *plane* is a definition composed of a concept (node) and its relations (edges).

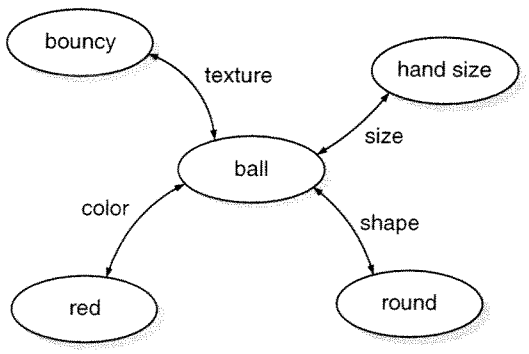

Figure 5.1: A semantic network. Nodes represent *concepts*, edges *relations*.

From this perspective, concepts have *no* meaning in isolation, and only exhibit meaning when viewed relative to the other concepts to which they are connected by relational arcs. Structure is therefore everything in semantic

networks. Taken alone, a node is merely a syntactic token that happens to possess a convenient English label, yet from a computer's perspective, even this label is an arbitrary alphanumeric symbol. But taken collectively, the nodes exhibit a complex inter-relational structure that can be seen as meaningful. Long-term memory, for example, can be seen as a complex graph structure then in which ideas, events and experiences are all represented in this arcs and nodes fashion.

5.2.2 The Copycat Project

The Copycat project [193, 195, 275] is aimed at modeling cognitive-level behavior by simulating processes at the subcognitive but superneural level. Copycat is a computer program that can discover insightful analogies in a psychologically realistic way. Its behavior emerges as a statistical consequences of many small computational (stochastic) actions.

Analogies can well be illustrated graphically as shown in Figure 5.2. The first step consists in understanding and describing how A becomes B. The knowledge acquired is then used to find the matching answer for C.

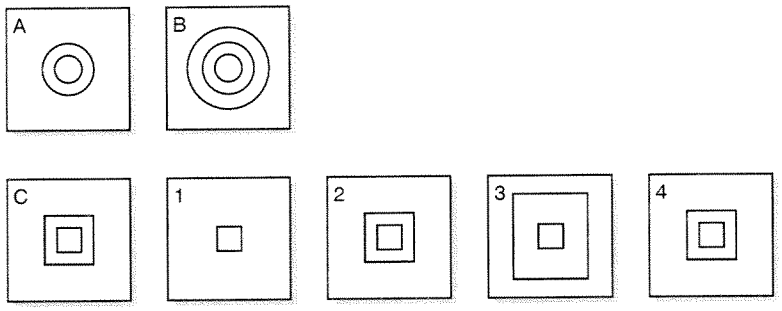

Figure 5.2: Analogy making: how C becomes the answer (knowing how A becomes B)?

The Copycat project does not deal with analogies between graphics but is concerned with analogy problems between letter strings, as for example:

$$\text{abc} : \text{abd} :: \text{ijk} : ? \tag{5.1}$$

If Copycat is told that the letter-string `abc` were changed to `abd`, and then asked to change the letter-string `ijk` "in the same way", Copycat — like most people — will propose the answer `ijl`, noting that `c` has changed into its alphabetic successor `d`.

The Copycat architecture is neither symbolic nor connectionist. It actually lies somewhere between these two extremes, although "[t]he use of parallel, stochastic processing mechanisms and the implementation of concepts as distributed and probabilistic entities in a network make Copycat

somewhat similar in spirit to certain connectionist systems" [195]. The main components of the architecture are the following:

- The *Slipnet*, a network of concepts (a, b, ..., z, letter, successor, predecessor, alphabetic-first, alphabetic-last, alphabetic position, left, right, direction, leftmost, rightmost, middle, string position, group, sameness group, successor group, predecessor group, group length, 1, 2, 3, sameness, opposite). The relevance of concepts (their activation) and the conceptual distance between them (the length of the links) change dynamically over the course of a run, depending on CopyCat's developing perspective on the given problem;
- The *Workspace* contains instances of various concepts from Slipnet combined into temporary perceptual structures.
- The *Coderack* is a list of small agents that can be stochastically called to carry out tasks in the Workspace.

It is important to note that Copycat does neither work with a "representation" of a problem created independently of the program's actions nor does it have a built-in "reasoning engine".

Hofstadter and his group have shown that a number of rich and complex analogies can be drawn in the Copycat system. They believe that the capability do see patterns and to draw analogies is very closely related to what we commonly call "intelligence". Their approach, however, presumes that "intelligence" is something algorithmic, which can be reproduced by machines. Whether this is true or not is an open an much discussed question.

5.3 Concepts, Mappings, and Mental Representations

5.3.1 Concepts

Concepts are among the most fundamental constructs in the theory of mind and properly contributed to the success of cognitive science. They are used to interpret our current experience by classifying it as being of a particular kind, and hence relating it to prior knowledge. Concepts are also fundamental to reasoning for both machines and people. In traditional AI, they are the symbolic elements from which the knowledge representation is built in order to provide the "expertise". Concepts are enlaced with almost all aspects of cognition and with a vast range of theories. It is therefore not surprising that they also rise many controversies. This usually already starts with the definition of what a concept is, whether they are "objects" or some other "behavioral abilities". Asking a psychologist, a philosopher, or a linguist

what a concept is is much like asking a physicist what mass is: no answer can be given in isolation, i.e., the notion of a concept cannot be explained without at the same time sketching the background against which it is set.

The best known and most discussed concepts are probably *lexical concepts* such as DOG, FLOWER, etc. They usually correspond to lexical items in natural languages. Margolis and Laurence [237] mention two other points in terminology. Concepts are said to be *primitive* if they lack structure, on the other hand, concepts are *complex* if they are not primitive. Primitive concepts are sometimes also called *atomic concepts* or *features*. They also honestly admit the following: "What exactly it means to say that a concept has, or lacks, structure, is another matter". For more details, see [237].

A concept might be basically seen as a reaction to, or a development of what is commonly known as the *Classical Theory of Concepts*. The definition according to Margolis and Laurence [237] is as following:

Definition 5.3.1 (Concepts: The Classical Theory)
"Most concepts (esp. lexical concepts) are structured mental representations that encore a set of necessary and sufficient conditions for their application, if possible, in sensory or perceptual terms" [237, p. 10]. ■

The classical model assumes that concepts are clearly defined by a conjunction of singly necessary and jointly sufficient attributes [22]. During the 1970s, the Classical Theory was became a first serious alternative, the *Prototype Theory*, which was developed to a large extent to accommodate the psychological data that had proved to be damaging to the Classical Theory (for a summary of the critics see [237, p. 27]).

Definition 5.3.2 (Concepts: The Prototype Theory)
"Most concepts (esp. lexical concepts) are structured mental representations that encode the properties that objects in their extension tend to possess" [237, p. 31]. ■

In this theory, concepts are represented by a prototype with all the most common attributes of the category, which includes all instances sufficiently similar to this prototype. Where the Classical Theory characterized sufficient conditions for concept application in terms of the satisfaction of all of a concept's features, the Prototype Theory application is a matter of satisfying a sufficient number of features, where some may be weighted more significantly than others [237]. For example, the instance BIRD is composed of such features as FLIES, SINGS, NESTS IN TREES, LAYS EGGS, and so on. As Margolis and Laurence state, on the Prototype Theory, robins are an

extension of BIRD because they tend to have all of the corresponding properties. However, BIRD also applies to ostriches because even though ostriches don't have all of these properties, they have enough of them. Prototype theorists have also developed a number of different measures for determining the similarity of concepts or items. For more detail see [237, p. 30] or [382].

There are three other models that are usually considered:

- the *Theory-Theory*,
- the *Neoclassical Theory*, and
- the *Conceptual Atomism*.

The interested reader is referred to the comprehensive volume edited by Margolis and Laurence [237]. A slightly different classification of the models is provided in [22].

5.3.2 Mappings and Mental Spaces

In a mathematical sense, a mapping is a correspondence between two sets that assigns to each element in the first a counterpart in the second. For example, the set $A = \{a, b, c\}$ might be associated with the set $B = \{1, 2, 3\}$ via the following mapping: $a \rightarrow 3, b \rightarrow 1$, and $c \rightarrow 2$. Mappings as a cognitive faculty can be consider as the faculty of producing, transferring, and processing of meaning. Mappings operate to build and link *mental spaces* hat have been set up during a discourse.

Mental space theory — which basically originated in the work of Fauconnier [127] — focuses on the subtle relationship among elements in the various mental models that speakers construct. The superior area of research is that of *cognitive linguistics*, which cannot be described as a single but rather as a set of distinct theories. Various theoretical frameworks exist that are to a large degree mutually compatible.

According to Fauconnier, *mental spaces* are partial structures that proliferate when we think and talk, allowing a fine-grained partitioning of our discourse and knowledge structures. For example (from [128]), in saying *Liz thinks Richard is wonderful*, we build a space for Liz's reported beliefs, with minimal explicit structure corresponding to Richard's being wonderful.

Or in other words, mental spaces are

> "[...] built up in any discourse according to the guidelines provided by the linguistic expressions. In the model, mental spaces will be represented as structured, incrementable sets— that is, sets with elements $(a, b, c, \dots)$ and relations holding between them $(R_1ab, R_2a, R_3cbf, \dots)$, such that new elements can be added to them and new relations established between their

> elements. (In a technical sense, an incrementable set is an ordered sequence of ordinary sets, but it will be convenient to speak of the mental space as being built up during ongoing discourse, rather than to refer to the corresponding sequence of sets.) [...] A partial ordering relation is defined on spaces. I shall call it *inclusion* and use the usual notation $\subset$, but unlike set inclusion, it carries no entailments for the elements within the spaces: $a \in MandM \subset N$ does not entail $a \in N$" [127, p. 16].

Expressions (typically linguistic expressions) that establish new spaces, elements within them, and relations that hold between the elements or that refer to already existing spaces are called *space builders*. The space-builder that builds space M is specified as SB_M. When a new space is created in a discourse by some space-builder SB_M, it is always connected to some parent space.

The connection between concepts and mental spaces is somehow vague and depends on the paradigms used. Meaning might be characterized as conceptualization in the sense that the meaning of an expression is given by the concepts that are *activated* in the mind. Mental spaces — which might be considered as *activated* neural assemblies — are something much more dynamical that concepts that develop and change slowly. On the other hand, mental spaces usually make use of or directly are concepts. For example, a mental space might contain the concepts HAIR and BRUSH. For, Pereira and Cardoso [324], seen from the point of view of AI researchers, a mental space is representable as a *semantic network*, a *frame*, a *case*, or any other symbolic knowledge structure that gathers a set of inter-related concepts towards a specific situation.

Example 5.3.1 (Mental Space and Connections)
Let us consider a example taken from [128]. Assume that we are having a conversation about Romeo and Juliet an that the statement "Maybe Romeo is in love with Juliet" *is made. This sentence would bring in a frame "x in love with y" with two roles highlighted, i.e., (1) the lover x and (2) the loved y. The word "maybe" is a space builder that will set up a new space relative to a base space. The base space contains the elements a and b associated with Romeo and Juliet. The new possibility space creates counterparts a' and b' for a and b which will be identified by the names Romeo and Juliet. Furthermore, the new space is structured by the frame "x in love with y" whose roles are filled by the elements a' and b'. Figure 5.3 illustrates the representation usually used. Frames are denoted in capitalized words. The dashed line indicates that M is set up* relative *to B. The boxes represent internal structure of the spaces next to them. Structure from the parent space is transferred to the new space*

by default and results in associating a′ with Romeo and b′ with Juliet and also with other background structure for their counterparts in a and b in B [128].

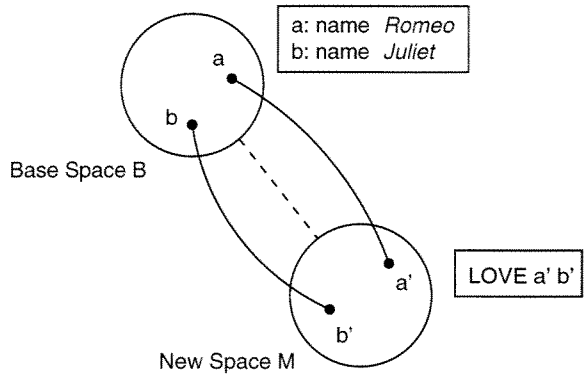

Figure 5.3: Illustration of the space creation in relation with the sentence "Maybe Romeo is in love with Juliet". Frames are denoted in capitalized words. The dashed line indicates that M is set up *relative* to B. The boxes represent internal structure of the spaces next to them. Redrawn from [128].

■

For more information about mappings and mental spaces, the interested reader is referred to [127, 128]. For the sake of completeness, it should also be mentioned that conceptualism has been challenged in recent years. An increasing number of philosophers have argued for the indispensability of nonconceptual content based on perceptual, emotional, and qualitative experiences, informational and computational states, memory, and practical knowledge (see for example [175] for more details).

5.4 Conceptual Integration and Blending

Human beings have developed an unprecedented ability to invent new concepts and meanings, to make discoveries, and to assemble new and dynamic mental patterns. The mechanisms of concept acquisition are probably among the most fundamental processes of cognition. Concepts allow to categorize a potentially unlimited number of objects and events with limited resources. Our ancestors gained the superiority that allowed to develop art, science, religion, culture, and language through the evolution of the mental capacity for *conceptual integration* or *blending*[1] [130]. Fauconnier

[1] Turner's website on blending and conceptual integration: `http://blending.stanford.edu`.

and Turner are the first to consider the process of blending as central, uniform, and pervasive, although many others considered blends—as a minor matter—before them. Lakeoff and Núñez [230], for example, claimed that blends are important in scientific creativity.

The main problem of blending is that it is difficult the be considered as a real theory because a formal approach is missing or is at least very vague. There are many examples that are not straightforward to explain with the general theory—that is not falsifiable in addition. There is no method that allows to say whether something is a blend or not. The research community is, however, rapidly growing and many efforts have already been made to propose a formal theory. Fore more details about computational blending, see Section 5.5.

The two-space model of metaphor has been a corner-stone of the metaphor field since Aristotle. The classic theories consider a metaphor as an interaction between two conceptual spaces, i.e., between a space that describes the metaphor (commonly termed *target*, *tenor*, or *topic*) and between the space that provides the description (termed *vehicle* or *source*). Fauconnier and Turner's blending framework is an elaboration of the classical metaphor theory. It augments the traditional input spaces with two new spaces (see later for more details): (1) the *generic space* that captures common background knowledge and that unites the inputs, and (2) the *blend space* that contains novel and emergent conceptual products of the integration.

Although blending is only a subset of all the ways mental things can be put together, it is a general cognitive operation that serves a variety of purposes. It seems to be at the root of the cognitively modern mind, is not reserved for special purposes, has fascinating dynamics, is supple and active in the moment of thinking, is not costly, and plays a crucial role in how we think and live. Blending operates almost invisibly to consciousness, it choreographs vast networks of conceptual meaning, yielding cognitive products, which, at the conscious level, appear simple. Conceptual blends themselves are repeatedly blended and reblended by people and their cultures to create the rich fabric of the way we live. The theory of conceptual blending has been applied in cognitive neuroscience, cognitive science, psychology, linguistics, music theory, poetics, mathematics, divinity, semiotics, theory of art, psychotherapy, artificial intelligence, political science, discourse analysis, philosophy, anthropology, and the study of gesture and of material culture.

Blends arise in networks of mental spaces. "Mental spaces are small conceptual packets constructed as we think and talk, for purposes of local understanding and action. Mental spaces are very partial assemblies containing elements, and structured by frames and cognitive models. They are interconnected and can be modified as thought and discourse unfold." [129]. Mental spaces are usually used to model dynamical mappings in thought

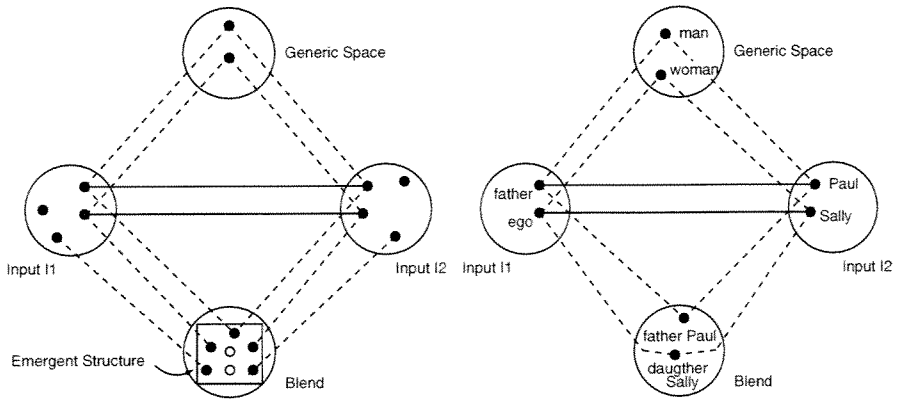

Figure 5.4: The basic diagram (on the left) illustrating the central features of conceptual integration [130]: The circles represent mental spaces, solid dots represent entities in each domain, solid lines indicate the matching (i.e., a *relation*) and cross-space mapping between the inputs, the dotted lines indicate connections (i.e., a *mapping*) between inputs and either generic or blended spaces, and the solid square in the blended space represents emergent structure. The generic space contains what inputs have in common. On the right, a simplex network where one input contains a frame with no values, the other input contains unframed elements. Inputs are matched by a frame-to-values connection.

and language.

Figure 5.4 (on the left) shows four mental spaces. The circles represent mental spaces, solid lines indicate the matching (i.e., a *relation*) and cross-space mapping between the inputs, the dotted lines indicate connections (i.e., a *mapping*) between inputs and either *generic* or *blended* spaces, and the solid square in the blended space represents emergent structure. At any moment in the construction of the network, the structure that inputs seem to share is captured in a generic space ("man" and woman"). In blending, structure from two input mental spaces is projected in a new space, the *blend*. The blend contains emergent structure coherently projected (by *composition*, *completion*, or *elaboration*) that cannot be found in the inputs ("father Paul" and "daughter Sally"). Note that not all elements from relations from the inputs are necessarily projected to the blend. The *generic space* contains the low-level conceptual structures that serve to mediate between the contents of the input spaces and to structurally reconcile them.

Let us take a look at a simple example (from [129]): "They are digging their financial grave", illustrates a projection from one input of grave digging and another of financial investment. In "This surgeon is a butcher", there is also projection from two input spaces. In both cases, the central inference is constructed in the blend. The projection is selective. Through completion

and elaboration, the blend develops structure not provided by the inputs. Inferences, arguments, and ideas developed in the blend can have effect in cognition, leading us to modify the initial inputs and to change our view of the corresponding situations.

Blending is essentially nondeterministic: given the nature of mental operation, it can not be predicted that from two inputs a certain blend must result or that a specific blend must arise at a given place and time.

Finally, it should also be mentioned that blending a purely "symbolistic' approach. According to Harnad [180], a symbol system is:

1. a set of arbitrary "physical token" scratches on paper, holes on a tape, events in a digital computer, etc. that are
2. manipulated on the basis of "explicit rules" that are
3. likewise physical tokens and strings. The rule-governed symbol-token manipulation is based
4. purely on the shape of the symbol tokens (not their "meaning"), i.e., it it purely syntactic, and consists of
5. "rulefully combining" and recombining symbol tokens. There are
6. primitive atomic symbol tokens and
7. composite symbol-token strings. The entire system and all its parts—the atomic tokens, the composite tokens, the syntactic manipulations both actual and possible and the rules—are all
8. "semantically interpretable:" The syntax can be systematically assigned a meaning, e.g., as standing for objects, as describing states of affairs.

Harnad reconstructed these points from Newell [289], Pylyshyn [328], Fodor [135] and the classical work of Von Neumann, Turing, Gödel, Church, etc. (see Kleene [220]) on the foundations of computation.

The classical symbolistic approach — essentially based on the notion of knowledge [290], usually represented as symbols, that describe goals and actions — known from the field of artificial intelligence has been extensively criticized on several grounds and a number of fundamental problems have been identified [440]:

- the *frame problem* [142];
- the *symbol grounding problem* [180];
- the *frame of reference problem* [72], and

- the *problem of situatedness* [396].

Since blending is a symbolistic approach too, it basically has to deal with the same problems. I will however not enter into this difficulty as it is somehow secondary for our proposes.

In the following two sections, the *constructing*, the *governing*, and the *optimality principles* of blending shall be briefly presented. These principles are more descriptive than explanatory and a lot of practical questions—that will be partially addressed in Section 5.5—remain open.

5.4.1 Constructing Principles

The constructive principles of blending—that allow to obtain emergent structure—are described in [130] as follows:

- **Matching and counterpart connections:** Look for a partial cross-space mapping between counterpart connections of the two input spaces.
- **Generic space:** Look for elements and structure that both input spaces have in common. "[P]owerful generic spaces can themselves become conventional and serve as resources to be drawn on in attempts to build new cross-space mappings in new integration networks" [129]. The blended space will contain generic structure captured in the generic space, but it will also contain emergent structure not present in the inputs.
- **Blending:** In order to obtain emergent structure in the blend, the following constructive mechanisms are used.
  - **Selective projection:** Not all structure from the inputs is projected to the blend.
  - **Composition:** The composition of elements makes relations available that did not exist in the two input spaces.
  - **Completion:** After composition, the structure obtained is completed with additional structure while existing structure remains.
  - **Elaboration:** Finally, (on-line) elaboration develops the blend by "running the blend", i.e., an imaginative simulation is executed that provides new elements. Elaboration consists in cognitive work performed within the blend, according to its own emergent logic. It is generally assumed that cognitive systems construct models of the problem space that are the "run" or manipulated to produce the desired outputs.

The order of these steps may be changed and several interactions of the process may be necessary.

5.4.2 Governing and Optimality Principles

Optimality and governing principles are used to make "good" and "well-formed" blends and characterize strategies for optimizing emergent structure. They frequently compete with each other and that results in a variety of lower-level optimality pressures for constructing the blend, so one should not expect to observe them in any given integration in a perfect way. "The constructive and governing principles have the effect of creating blended spaces at human scale" [130].

The following governing and optimality principles are described in "The Way we Think":

- **Compression**: Relations can be compressed into a tighter version of itself, several relations can be compressed into one relation, or new compressed relations might be created from scratch. One of the most simple kinds of compression is scaling down. Another frequently used compression technique consists in shortening the causal chain between cause and effect.

- **Topology principle**: Makes sure that the semantics of the integration are valid by ensuring that those corresponding elements that are blended together relate to the other elements of their spaces in a similar fashion, i.e, it ensures that "like" is blended with "like". In other words, the blend and the inputs should be set up so that useful topology in the inputs and their outer-space relations (i.e, relations between the spaces) is reflected by inner-space relations (i.e, relations within the spaces) in the blend.

- **Web**: Ensures that the other constraints (especially the *Integration* constraint) do not sever the links between newly blended elements and their original inputs, i.e., manipulating the blend as a unit must maintain the web of appropriate connections to the input spaces easily and without additional surveillance. The web and the topology principle are distinct and usually compete. The topology principle tries to maximize topology between an input and the blend, whereas the web principle tries to maintain good and appropriate connections between the spaces.

- **Unpacking**: This constraint states that one who comprehends the blended result of an integration should also be able to reconstruct the network of spaces that were on its origin, i.e., the blend itself should prompt for the reconstruction of the entire network.

- **Relevance**: An element in the blend should have relevance, including relevance for establishing links to other spaces and for running the blend. Conversely, an outer-space relation between the inputs that is important for the purpose of the network should have a corresponding compression in the blend.

- **Pattern completion principle**: Complete elements in the blend by using existing integrated patterns as additional inputs. Other things being equal, use a completing frame that has relations that can be the compressed versions of the important outer-space vital relations between the inputs.

- **Maximization of vital relations**: Vital relations in the network should be maximized. In particular, maximize the vital relations in the blended space and reflect them in outer-space vital relations.

- **Intensification of vital relations**: Vital relations should be intensified.

- **Integration**: The inputs often have rather different topologies. The goal of integration is to achieve an integrated blend by making the right selections and adjustments, i.e., that the blended elements are readily manipulated as single conceptual units.

5.4.3 Emergent Structure

The principal power of blends lies in their *emergent structure*, i.e., structure that was not available in the input spaces. This structure is revealed by "running the blend", more often called *elaboration*. Elaboration consists in cognitive work performed within the blend, according to its own emergent logic. Elaboration is nonpredictive, i.e., it is impossible to know in advance where it will lead.

As Turner and Fauconnier claim, blends are not just conceptuals constructions but they illustrate a deep property in relation to creativity. Blends are genuine domains of mental exploration and "running" them can lead to deep discoveries that were all but anticipated in the initial inputs. Fauconnier writes:

> "Mathematicians did not start out with the intuition that there were non-Euclidian geometries waiting to be formalized. On the contrary, they were firmly convinced that geometry was inherently Euclidian, and the purpose of Saccheri's counterfactual blend was to establish this rigorously through reductio. Instead, what happened was that the blend took on a life of its own: It contained what its creator had not foreseen, "hyperbolic

geometry," a radically novel conception of space, which Einstein and others would later apply to the physical universe" [128].

5.4.4 Examples

In this section, two examples will be considered in order to better illustrate how blending works. For many more examples, see [130].

Example 5.4.1 (Reductio Blend)
Let's consider an example taken from [128]. A common method for proof is the reductio ad absurdum*: if one seeks to prove "not P", he sets up the hypothesis "P" together with known axioms and theorems and operates in this mathematical context till he reaches a contradiction. From that contradiction one can draw the conclusion that "not P" is true. Figure 5.5 illustrates the* reductio ad absurdum *method to proof the theorem* $(A \Rightarrow B) \Rightarrow (nonB \Rightarrow nonA)$.

1	A => B	hypothesis
2	nonB	hypothesis
3	A => B	1, a fortiori
4	A	hypothesis
5	A => B	1, a fortiori
6	B	4, 5, modus ponens
7	nonB	2, a fortiori
8	⊥	6, 7, contradiction
9	nonA	reductio ad absurdum
10	nonB => non A	
11	(A => B) => (nonB => nonA)	

Figure 5.5: Proving the theorem $(A \Rightarrow B) \Rightarrow (nonB \Rightarrow nonA)$ by means of the *reductio ad absurdum* method as presented in [471, p. 123].

The same reasoning can be found in a counterfactual *blend as seen in Figure 5.6. Fauconnier writes:*

> *"In Input* 1*, we have the axioms and relevant theorems of the mathematical system being used. In that space, P is not established; its status is unknown. In Input* 2*, P is true, but the axioms leading to P are not known. In the blend, known axioms and theorems are imported from Input* 1*, along with P from Input* 2*. As in the general case of blending, we can "run the blend" to uncover its structure. In the case of reductio, the goal is to reveal the mathematical inconsistency of*

B by producing a contradiction in B. The blend is therefore "fictive": It does not correspond to a viable mathematical situation. The point of running the blend is to show that it cannot match Input 1*, and therefore that P does not hold in Input* 1*. And the reason it cannot match is that Input* 1 *is assumed to contain no contradiction" [128, p. 165].*

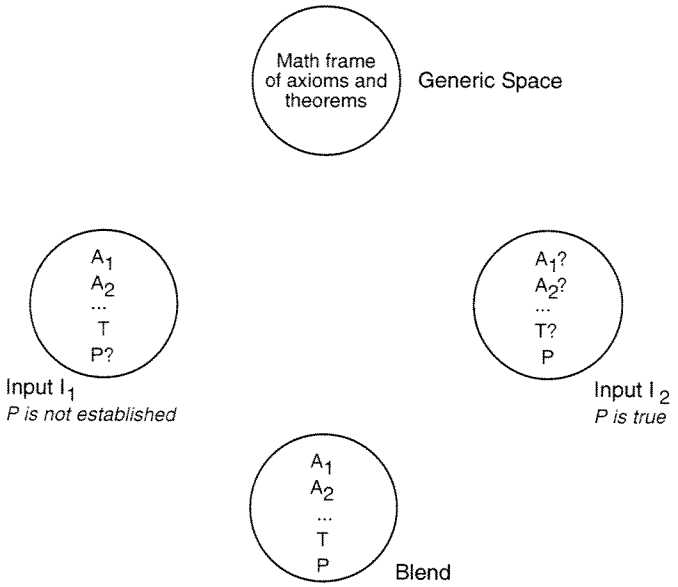

Figure 5.6: The configuration of the reductio ad absurdum blend. Redrawn from [128].

■

Example 5.4.2 (Complex Number Blend)
Let us consider another example given by Fauconnier and Turner in [129, p. 146–149] on complex numbers.

Complex numbers—first introduced in connection with explicit formulas for the roots of cubic polynomials—are an extension of the real numbers: the real number line is enlarged to get the complex number plane. The complex numbers are the field C of numbers of the form $x+iy$*, where* x *and* y *are real numbers and* i *is the imaginary unit equal to the square root of* -1*,* $\sqrt{-1}$ *(see Figure 5.7). When a single letter* $z = x+iy$ *is used to denote a complex number, it is sometimes called an* affix*. In component notation,* $z = x + iy$ *can be written* (x, y)*. The field of complex numbers includes the field of real numbers as a subfield.*

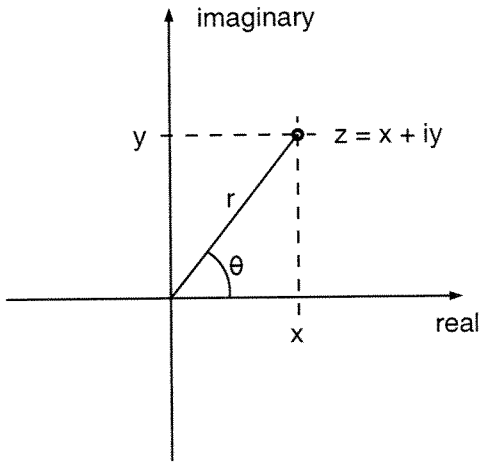

Figure 5.7: Complex numbers represented as a point in a two-dimensional space.

Figure 5.8 shows the blend of complex numbers. There is an initial cross-space mapping of numbers to geometric space, a generic space, and a projection of both inputs to the blend. In the blend, the numbers are fused with the geometric points and emergent structure appeared. In the blend, but not in the original input space, it is possible for an element to be simultaneously a number and a geometric point specified by cartesian coordinates (x, y) and polar coordinates (r, θ). Completion made available arguments (θ) and magnitudes $(r = |z|)$, whereas elaboration provided multiplication and addition reconstructed as operations in vectors. As Fauconnier and Turner state: "The blend takes an realist interpretation on mathematics. It constitutes a new and richer way to understand numbers and space. [...] It highlights the deep difference between naming and conceptualizing; adding expressions like $\sqrt{-1}$ to the domain of numbers, and calling them numbers, is not enough to make them numbers conceptually, even when they fit a consistent model."

■

5.4.5 Wrap-Up

As with any new theory, there are several problems and open questions. From the point of view of computer scientists, the most intriguing point is probable the following: "[c]onceptual integration is not a compositional algorithmic process and cannot be modeled as such for even the most rudimentary cases" [129]. Furthermore, Faconnier and Turner state "[...] despite decades of focused effort, it has been proven to be impossible to reduce semantics and other important forms of thought to symbolic logic. Because

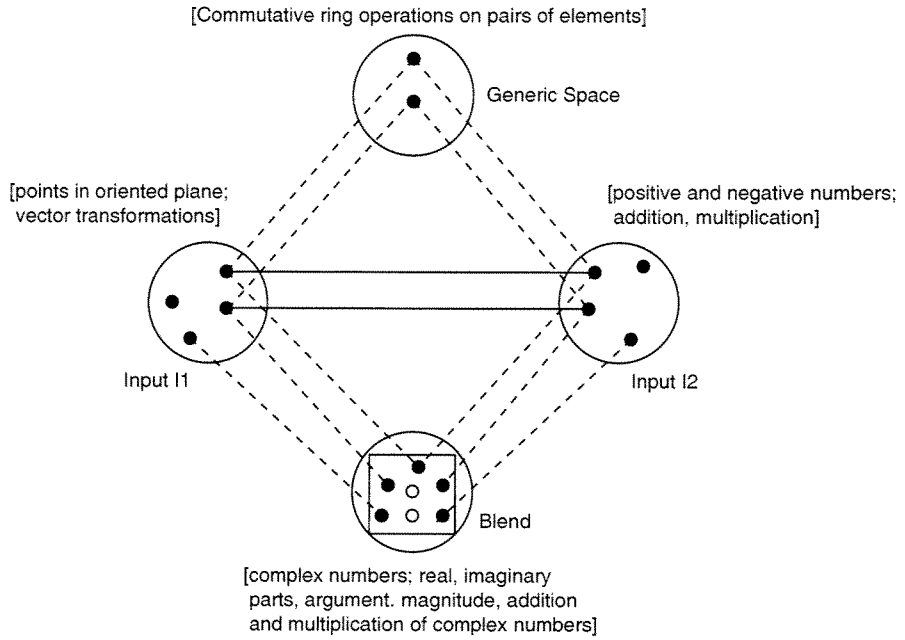

Figure 5.8: Complex number blend. Source: [129].

the construction of meaning requires many kinds of integration networks in addition to simplex networks, a great deal of semantics falls outside the realm of symbolic logic" [130].

This is, at least for me as an advocate of the strong Church-Turing thesis, very hard to believe. As the work of Pereira and Cardoso and others have shown (see next section), it clearly looks as blending *can* be modeled by an algorithmic process. And actually the point is rather that blending *has* to be an algorithmic process if people do not want to become it a kind of a "mystical" and highly non-scientific thing.

A second open question is how feedback from the environment comes in and helps to decide whether a blend is good or bad. I can hardly believe that the governing and optimality principles are sufficient for this. There must be other additional mechanisms which allow to decide (consciously or not) whether a newly created blend is useful or not.

Interestingly, blending has in some ways similarities, at least at a first glance, to the *adaptive cultural model* proposed by Kennedy [216]. The model has three main parts:

1. evaluate,
2. compare, and
3. imitate.

These parts might be opposed to the three constructing principles. Besides this rather afar taken similarity with the adaptive cultural model, which of course has nothing to do with blending otherwise, conceptual integration is an original approach without one's own kind.

Blending is an emergent field of research with a very active community, its center of gravity clearly lies around Gilles Fauconnier at the University of California San Diego (UCSD), Mark Turner at the Center for Advanced Study in the Behavioral Sciences at Stanford University and at the University of Maryland, and also at the University of Southern Denmark in the Mental Spaces Lab[2]. Recently, an special effort at UCSD has also been made to investigate the neural basis of blending (see also [240]). As far as I have been told, the investigations also partly involve the lab of the distinguished Vilayanur S. Ramachandran, director of the Center for Brain and Cognition and professor with the Psychology Department and the Neurosciences Program at UCSD.

5.5 Computational Blending: A Field Review

5.5.1 Introduction

Computer programs that learn and automatic programming will very probably be among the most important areas in computer science for the next decade. *Machine learning (ML)*, a term coined in 1959 by Samuel [352], is the field of research that is concerned with the study of computer programs that improve automatically through experience (see also [277]). Since the 1940s, computer scientists tried to give computers this ability. Early work came from McCulloch and Pitts [261] and from Turing [426] (see also [408]). In general, the degree of success was varying and rather modest. Today, computers that learn remain a key issue to machine intelligence, and it seems that we are still just scratching the surface of this problem.

In general, most high-level cognitive operations are partly quite well known, but nobody exactly knows how they emerge from the interplay of billions of neurons. On the other hand, much controversy and active discussions are on whether the cognitive system is functionally decomposable or not (see for example [31]) and whether the human consciousness may be modeled by a computer or not. At least so far, no grammatical system has ever successfully captured any complex natural system. Fauconnier and Turner likewise argue that conceptual blending is a non-algorithmic process that cannot even be modeled for the simplest cases. This was challenged by various people already: the first to propose a computational model of blending seem to be—to the best of my knowledge—Veale and O'Donoghue [437].

[2] `http://www.humaniora.sdu.dk/mentalspaces`.

They use Veale's metaphor integration network *Sapper* (see Section 5.5.2) to implement conceptual blending. A much more formal approach on blending came from Joseph Goguen who used algebraic semiotics [163] (see Section 5.5.3). Similar to classical model theory, semiotics also studies the relations between different models of a given theory. In [205], Indurkhya takes a somehow different approach and reverses the standpoint (see also Figure 5.9): different theories of the same model are considered. This is similar to different cultures and languages that describe a given situation, environment, and experience. Indurkhya formalizes the environment—the model—as an algebra, i.e., as a set of objects with a set of operators over it. There are no predicates in this algebra since he believes that operators are more primitive that relations (according to Piaget [325]) and that all relations can be broken down into a sequence of operators.

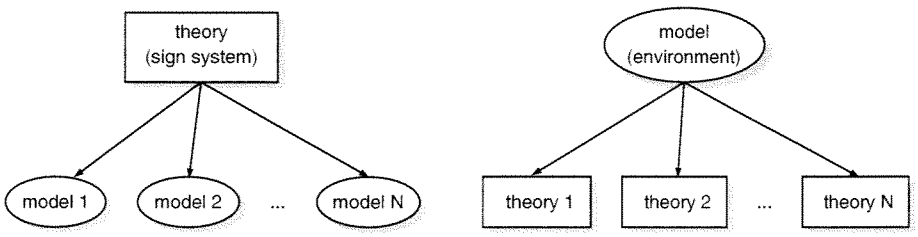

Figure 5.9: Focus on multiple models of a theory (left) and focus on multiple theories of a model. Redrawn from [205].

5.5.2 The Sapper Model

Sapper [437, 438] is a hybrid symbolic connectionist and parallel adaptive model that views the interpretation of novel metaphors as a process of connectionist bridge-building. Sapper originated in the PhD thesis[3] of Tony Veale [436] where he investigated metaphors, memories, analogies, and meaning. Metaphors form an important aspect of human creativity and Veale started from the hypothesis that computational perspectives on metaphor might shed light on the broader issues of creative cognition.

Sapper employs a top-down rule-based structure recognition with a bottom-up spreading activation approach to metaphor comprehension. The model attempts to recognize cross-domain bridges that will connect the *tenor* and *vehicle* schemata of a metaphor in a systematic manner (see Figure 5.10). The structural and symbolic component lays down dormant bridges between concepts that, according to local criteria, seem possible future analogues of each other. Whenever activation waves from the tenor and vehicle concept matriarches cross-over at such a bridge-point, the

[3] Also available online: `http://www.compapp.dcu.ie/~tonyv/thesis.html`

end-points of this bridge are considered to be tentative mappings in the final interpretation.

In the Sapper model, activation is considered to travel in waves, each wave-form possessing an amplitude and a unique signature frequency [436]. In this view of memory, each localist concept node is considered to possess a unique resonant frequency, which is used to modulate any activation waves that pass through this node. The signature frequency of an activation wave thus combines the resonant frequency of the source matriarch node of the wave, representing either the tenor or vehicle concept of the metaphor, as well as the resonant frequencies of any other nodes encountered en-route by the wave.

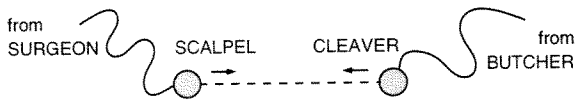

Figure 5.10: Illustration of a dormant linkage between the two concepts SCALPEL and CLEAVER. The linkage provides a plausible match hypothesis when it becomes a cross-over path or competing activation waves from the SURGEON (*tenor*) and BUTCHER (vehicle) concept nodes. Redrawn from [438].

Sapper offers a cognitively appealing view of metaphor as a creative force that invents, rather than simply observes, new associations between concepts. The Sapper model is written in Prolog and can be obtained from Veale's web-site: `http://www.compapp.dcu.ie/~tonyv/sapper.tar.gz`. For a comprehensive description, the interested reader is referred to [438] or [436].

5.5.3 Algebraic Semiotics

The design of a user interface is a non-trivial task and involves mappings, concepts, analogies, and representations. "A user interface can be considered as a representation of the underlying functionality to which it provides access, and thus user interface design can be considered a craft of constructing such representations, where both the interface and the underlying functionality are considered as (structured) sign systems" [163]. Naturally, user inferfaces also make intensive use of symbols and signs[4] This has motivated the development of a calculus, called *algebraic semiotics* that allows to combine signs, sign systems, and representations. Fauconnier and Turner's conceptual spaces are also sign systems that usually only make use of constants (and not of constructors) and relations between them. In the context of algebraic semiotics, blending can be considered as a mode of composition

[4] *Semiotics*: the study of signs.

that combines two input spaces (i.e., sign systems) to obtain a new space, the blend, that contains new and emergent meaning (i.e, signs).

Figure 5.11 the diagram of the basic kind of blend. In terms of algebraic semiotics, a blend B of sign systems I_1 and I_2 over G is defined by the morphism $I_1 \rightarrow B$, $I_2 \rightarrow B$, and $G \rightarrow B$ (Fauconnier and Turner do not make us of this morphism).

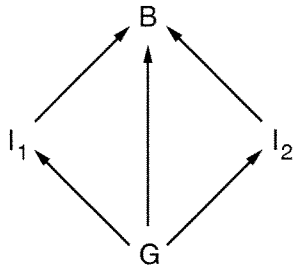

Figure 5.11: Diagram that visualizes the basic kind of blend. Source: [163]. Note that this diagram is "upside down" compared to Fauconnier and Turner's notion.

5.5.4 Pereira & Cardoso's Dr. Divago

The big challenge of computational blending and creativity is the creation of new concepts. Fauconnier and Turner's blending is an elegant proposal for a creative process in cognition, but if lacks a formalism across several aspects which makes it somehow difficult to consider it as a verifiable and falsifiable scientific theory.

Pereira and Cardoso are the creators of the *Divago* (also *Dr. Divago*) system [319, 322] and are — to the best of my knowledge — the first and few people who are working on computational blending. The main goal of Dr. Divago is to model divergent thought in problem solving by proposing solutions outside the domain focus.

Their first model (part of Divago), presented in [320], is called the *blender*. Its goal is to blend two different domains into a new, third domain. A domain is considered as a subtype auf Fauconnier's mental space [127] and consists of two kinds of knowledge (see [320]): (1) *Domain Theory* and (2) *Domain Instances*. The definitions provided are pretty formal and shall not be reproduced here.

The blender does not make use of optimality principles, and a newly generated blend is therefore not at all guaranteed to make sense or to be consistent. This problem was improved later: in [324], Pereira and Cardoso present a formalization of the optimality principles for their computational approach, which makes us of a specific type of mental spaces, called *domain*, and which are close to the *domain knowledge* in AI. They also propose to

reduce the eight optimality principles (Section 5.4.2) to seven because the "web" principle is not independent. A further reduction might even reduce them to five only.

The Divago architecture as described in [323] has four main modules:

1. the *Knowledge Base*,
2. the *Mapper*,
3. the *Factory*, and
4. the *Constraints Module*.

The knowledge base represents the concepts in a classical and purely symbolic way (as opposed to a sub-submolic representation as used in neural network for example). The base is divided into several domains which have different types of knowledge: a *concept map*, a set of *instances*, a set of *rules*, a set of *integrity constraints*, or a set of frames (for more details, see [323]).

The mapper generates mappings between the concept maps of the input domains. This basically done by means of an algorithm which uses a spreading activation to search for the largest isomorphic pair of sub-graphs in the input domains. Thereby, two graphs are considered isomorphic if they have the same relational structure (i.e., arcs), independently of the concepts (i.e., nodes).

The factory searches for blends which fit the optimality principles. This search of course happens in a huge search space and its therefore very demanding in terms of computational power. Pereira and Cardoso use a genetic algorithm[5] to find a good blend that respects the optimality principles.

The constraints module, finally, is responsible for the implementation of the optimality pressures, which in turn are used as a fitness measure by the factory module.

The Divago architecture can be summarized as illustrated in Algorithm 9.

Algorithm 9 The Divago Blending Architecture

1: Build the concepts (KNOWLDEGE BASE).
2: Look for a mapping between the two input spaces (MAPPER).
3: **while** best or "good" solution not found **do**
4: Search for blends (FACTORY) that satisfy the optimality principles (CONSTRAINTS MODULE).
5: Apply crossover and mutation.
6: **end while**

[5]Note that they consider genetic algorithm itself as an example of a blend [321].

An experiment of their computational model is described in [321], where they blend "houses" and "boats" together. The experiment shows that it is absolutely crucial to provide sufficiently enough and precise definitions of the concepts "house" and "boat", which can be very tricky of course. The results are also represented visually in [321], but are not convincing, however, they clearly state that their goal was to test the capacity of generating new and different instances coherent with the blend, regardless of the aesthetical judgment. It seemed that the concepts of "house" and "boat" were too simplistic and a further problem was the absence of the optimality constraints in this experiment.

Further results with making use of the optimality pressures are presented in [323] and [324]. All experiments basically raised the fundamental question whether the computational model proposed agrees with Fauconnier and Turner's framework. However, since their framework is not formally defined, a straightforward comparison is impossible.

Pereira and Cardoso's work certainly presents the most complete computational blending framework, but a lot remains to be done. As they state, most problems are related to the optimality principles and it is all but straightforward to find an "optimum" between the novelty and the usefulness of the blends. The other fundamental problem is that the concepts have to be symbolically specified, which awakes problems (i.e., the grounding and the framing problem [142, 180]) of the field of classical artificial intelligence.

5.5.5 On the Creation of Novelty

The human mind is certainly one of the most creative systems one can imagine. "All forms of thought are creative in the sense that they produce new links, new configurations, and correspondingly, new meaning and novel conceptualization" [128]. A recent study [63] in the Journal of Personality and Social Psychology revealed that the brains of creative people appear to be more open to incoming stimuli from the surrounding environment. In addition, scientists have wondered for a long time why madness and creativity seem linked. The authors have in fact — probably for the first time — identified one of the biological bases of creativity and have moved towards cracking an age-old mystery: the relationship between genius, madness and the doors of perception.

Creativity and the creation of novelty in computers is a little different from humans and problems usually already begin with the definition of what "creativity" and "novelty" are. As Gross and McMullin put it, "[a]t a first glance, the problem of novelty creation in computers does not appear to be a problem at all" [166] since computer models are nothing but algorithms and will therefore—while correctly operating—not do anything else than the modeler has foreseen. The problem is probably more subtle and one has first to agree whether life is to be considered as an algorithmic process or

not. This question has not been answered to a generally and widely accepted degree so far. Further, we would also have provide a definition of "creativity" and "novelty" before we start talking about such ill-defined concepts.

Maybe the best known work on computational "creativity" (and analogy making) comes from Doug Hofstadter [193, 195, 275]. This project is described in more details in Section 5.2.2. The problem with logic and computation is—as already stated above—that the field is intuitively all but creative since everything is rule-based, algorithmic, foreseen, and dominated by early fixations in the design process.

It seems also obvious, that any credible artificial life-form would have to have the capability to create some kind of novelty. But the concept of novelty is rather ill-defined. It is often very hard to decided whether something is really genuinely novel or whether something is just a manifestation of something old. Worse, the concept of novelty is a relative concept: something is new only relative to something. However, the question might also been asked—without providing an answer however—whether there are really "new" things in the world.

In computer science, novelty might be defined as in Definitions 5.5.1 and 5.5.2.

Definition 5.5.1 (Computational Novelty I)
From a purely formal point of view, novelty in computer science might be defined as a system state that has not previously be occupied. ■

Definition 5.5.2 (Computational Novelty II)
From a human point of view, novelty in computer science should rather be defined as a new system state that has not previously be occupied and that has been reached by a nontrivial sequence of operations. ■

Most humans will not perceive any "novelty" when a new system state is reached by a more or less trivial and short sequence of operations. This is the main reason why Ronald *et al.* [344] propose "surprise" as an important element for the emergent creation of novelty. Obviously, it is basically also possible to modify the system state space — as illustrated in Example 5.5.1 — during operation, e.g, by adding or removing parameters. In such a case, one would however need an independent reference system (i.e., state space) that would allow to perceive the novelty.

Example 5.5.1 (State Space and Novelty)
Imagine the polynomial $y = ax^2 + bx + c$ *that contains three parameters* (a, b, c). *In terms of Definition 5.5.1, the parameter set* $(2, 5, -4)$ *represents a novelty compared to* $(4, -6, 0)$ *since they represent two different points in the state space.*

Depending on the problem, it might be more suitable to use a polynomial of higher degree such as for example $y = ax^3 + bx^2 + cx + d$. *In that case, the state space completely changes and one might say again that* $(-1, 4, 2, 3)$ *presents a novelty compared to* $(-2, 1, 4)$. ■

Novelty is also closely related to *emergence* (see [198] for an introduction). A behavior or structure is usually considered as emergent when the whole is more that the sum of its parts. Although there is nothing magic behind an emergent system, one might consider this as a form of creativity since new states in the systems' state-space are reached. One of the most illustrative examples is probably the *Game of Life* [39], where one can observe the "emergence" of gliders and other patters from a random initial configuration. However, the main objection raised by some people is the following: the rules of the Game of Life allow to create all those "emergent" structures from the beginning and everything is deterministic, so there's no emergence at all. Depending on the definition of emergence, this objection might of course be considered as valid. But the difficulties in defining emergence are reminiscent of the complications faced by early researchers in defining intelligence.

Nevertheless, several attempts to formalize emergence were recently made. Ronald *et al.* [344, 345] proposed an emergence test. Whereas the Turing Test [146, 424] is based on th human's incapacity at discerning human from machine, the emergence test centers on an observer's avowed incapacity (amazement) to reconcile his perception of an experiment in terms of a global world view with his awareness of the atomic nature of the elementary interactions.

The test basically consists in three conditions:

- **Design**. The system is described by a designer by means of local interactions between the components in a language $L1$.
- **Observation**. The global behavior of the system (as desired by the designer) is observed and described using a language $L2$.
- **Surprise**. The observer experiences surprise when the language of design $L1$ and the language of observation $L2$ are distinct, and the causal link between the elementary interactions programmed in $L1$ and the behaviors observed in $L2$ is non-obvious.

The test naturally depends on how big the surprise effect is, i.e., how easy it is for the observer to bridge the $L1$–$L2$ gap.

A further attempt to tame emergence came from Kubík and was also published in the Artificial Life journal [226].

Despite the difficulties encountered in defining and formalizing emergence, we know several issues that greatly help to improve the creation of novelty and "unexpected" appearances in computational systems:

- the set of primitives and the levels of abstraction should not be predetermined;
- make use of non-determinism and randomness;
- exploit "accidental changes";
- small changes at the bottom should allow for large changes on higher levels; and
- a sufficiently low-level description of the objects with a rich repertoire of operators should be used which allows to build higher-level structured objects and representations.

Naturally, the creative generation of novelty is also very closely related to problem solving in Nature and in computation. This can be well illustrated with an evolutionary algorithm. Evolutionary algorithms (EAs) (for an introduction see [28, 407] or Section 2.3) work with a population of individuals where each represents a solution in the solution space. Instead of a "blind search" strategy, the evolutionary algorithm makes use of genetic operators — such as crossover and mutation — to generate new and "better" individuals from the pool of already existing solutions. This is illustrated in Figure 5.12: the main question is how one can create a better solution — based on three parameters a, b, c, in this example? Without any background knowledge of the problem domain and the underlying structure of the fitness landscape, it is in principle impossible to generate a new and better set of parameters in the general case. The best strategy would simply be to randomly search for solutions.

Figure 5.13 shows a typical fitness landscape an evolutionary algorithm has to deal with. In that case, only two parameters are used. The z-axis represents the "fitness" (or quality) of the individuals in the parameter space. The goal is to maximize the fitness, i.e., to let at least one solution climb up to the highest "hill" in the fitness landscape.

Finally, I'd like to emphasize that, in analogy with the emergence of new types of proteins in real life, Gross and McMullin [166] identified artificial chemistries as potentially very promising for the perpetual creation of novelty. This was, of course, one of the reason why I have chosen artificial chemistries for my computational blending approach — the creation of novelty being one of the central aspects of Fauconnier and Turner's blending.

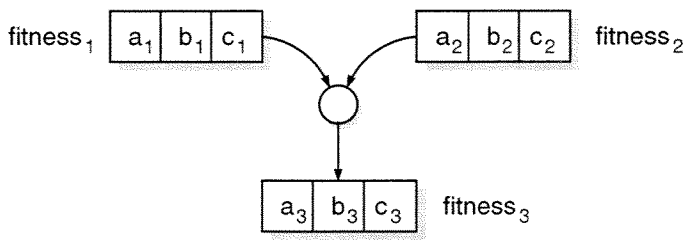

Figure 5.12: How can we create a "fitter" set of parameters from two different sets? Desired conditions: $\text{fitness}_1 < \text{fitness}_3$ and $\text{fitness}_2 < \text{fitness}_3$. In general, evolutionary algorithms make use of crossover and mutation in place of the abstract operator o.

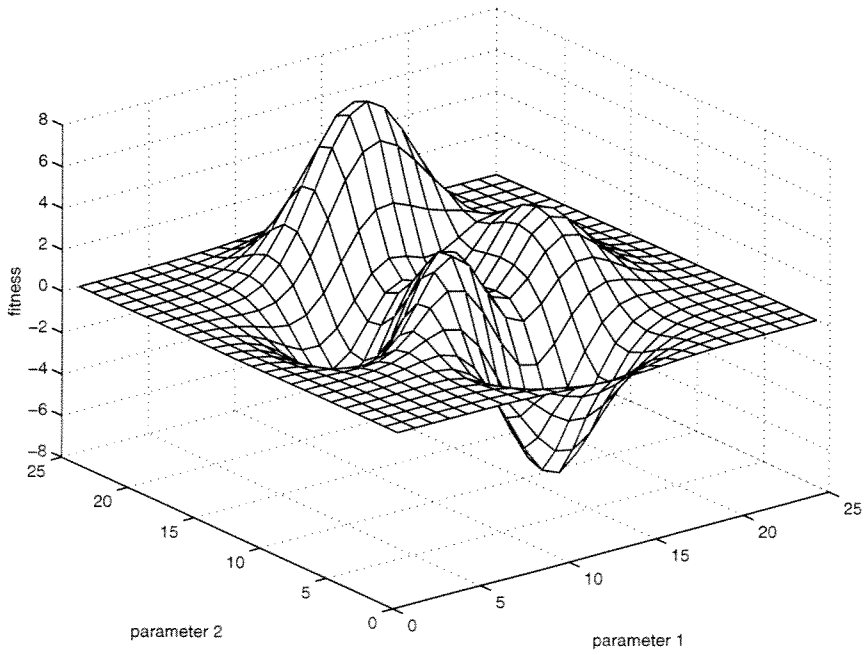

Figure 5.13: A fitness landscape as a function of two parameters.

5.6 C-Blending: A Computational and Cellular Approach

5.6.1 Introduction

One of the fundamental problems of membrane systems and artificial chemistries in general is the fact that there does not exist any universal approach which allows to choose a set of objects (chemicals) and rules (reactions) able to solve a given task. In their recent paper entitled "How to Program Artificial Chemistries", Busch and Banzhaf [56] illustrate an approach to use artificial chemistries to construct an automated theorem prover (ATP). Besides "creating" a chemistry by hand, one might for example also make use of evolutionary algorithms to find a possible set of rules and objects.

The main motivation for the development of the computational blending approach, entitled C-Blending, was to investigate the following questions:

- How might "novelty" be created in a "smart" way in a membrane system?
- Can Fauconnier and Turner's blending [130] be used in an elegant way to create "new" cells (in the sense of Paun's membrane systems) from two already existing cells? The newly created cell should contain novel, but also already existing elements.
- How can the constructing principles be realized?
- How can the governing and optimality principles be realized? Can they possibly be replaced by different or more general principles? Are they really necessary?
- How might blending be used as or inspire an organizing or optimizing principle for membrane systems?
- What are the benefits and drawbacks of such an approach?

While investigating the questions and implementing the concepts developed, it was a decided goal to keep everything as

- **simple**,
- **local**, and
- **parallel**

as possible, especially in view of a possible hardware realization. This, of course, required to compromise on several issues, but luckily, locality, simplicity, and parallelism are in general fully compatible with the concepts of an artificial chemistry, which is at the basis of our system. Nevertheless, it was for example unimaginable to faithfully implement Fauconnier and Turner's blending in a straightforward way since many elements, especially also the optimality principles, require to have a global view of the system. This is a serious issue and, as already stated somewhere above, I have my doubts regarding Fauconnier and Turner's optimality principles.

A word of caution: I mostly agree that the approach might seem rather weird at a first glance (and maybe even after a more profound investigation), but it was neither the goal to faithfully reproduce conceptual integration (or something else) nor to provide any biologically plausible model. All I was looking for was to investigate a novel and not less unconventional method to create "novelty" in membrane systems and how blending could inspire an organizing principle. As already stated in Section 1.1, various ways of taking into account Nature's "inventions" are imaginable. Trying to produce faithful copies of biological systems is usually doomed to failure because of the enormous complexity. The second possibility is to try to mimic only certain interesting features in order to build computational systems with novel properties and characteristics. I have basically chosen the latter, i.e., I didn't really care about biological, cognitive, or neuronal faithfulness, but was only interested to propose an approach with different characteristics and for different computational substrates.

In this purely theoretical section (for practical issues see Section 6) we shall present the C-Blending framework inspired by the principal ingredients of blending as presented in Section 5.4. My method somehow differs from the above presented computational blending approaches as it does *not* deal with concepts and mental spaces but with artificial chemistries and cells instead. The approach has been developed in view of later applications and extensions to P systems and to amorphous computers (see Chapters 6 and 7).

5.6.2 The Analogies Used

For C-Blending, the analogies — as presented in Table 5.1 — between concepts and the elements of membrane systems have been used. To consider a mental space as a membrane system is, as already said above, all but biologically plausible, but again, biological plausibility was *not* the goal. As we will see in the following, the chemical and cellular metaphors are quite natural with regards to the concept of blending.

From the computational point of view, a cell or membrane system is simply considered as a kind of "processor" which processes information. A multi-cellular system is therefore best compared with a massive parallel ma-

chine or with an agent-based architecture. From an abstract and dynamical systems point of view, a membrane system is a unit embodied within an "active" environment. The cell's output depends on its initial internal state S and on the history of its input vectors X_t. This is illustrated in Figure 5.14, where the membrane system is considered as a "black" box which simply processes information.

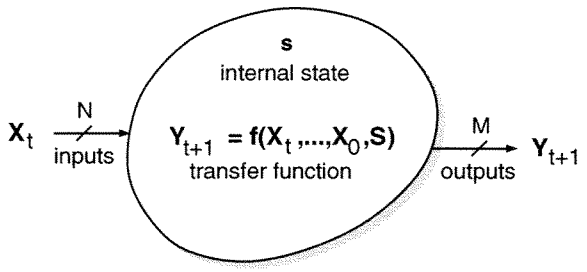

Figure 5.14: A membrane system seen as an information processing device. Its outputs depend on its initial internal state S and on the history of its input vectors X_t.

blending		membrane system
mental space	$\Longleftrightarrow$	membrane system
concept	$\Longleftrightarrow$	set of molecules and rules
object	$\Longleftrightarrow$	molecule
rule	$\Longleftrightarrow$	reaction
composition	$\Longleftrightarrow$	compose rules
completion	$\Longleftrightarrow$	complete rules
elaboration	$\Longleftrightarrow$	run the membrane system

Table 5.1: Analogies between blending and membrane systems used for C-Blending

When taking into account the analogies of Table 5.1, the C-Blending approach is summarized as depicted in Figure 5.15. Starting from two different membrane systems, a new cell will be "blended" which shall contain new structure and elements, but also elements from the two existing (mother) cells.

Note that we will not introduce a generic space as it is usually done in blending. The main reason for this choice was that a generic space would not have been very useful for our purpose.

As one can see, the three constructing principles of blending (i.e., composition, completion, and elaboration) are pretty straightforward to realize by means of artificial chemistries and membrane systems. Especially the elaboration step quite nicely corresponds to "running" a membrane system, i.e., applying the rules and let the system evolve.

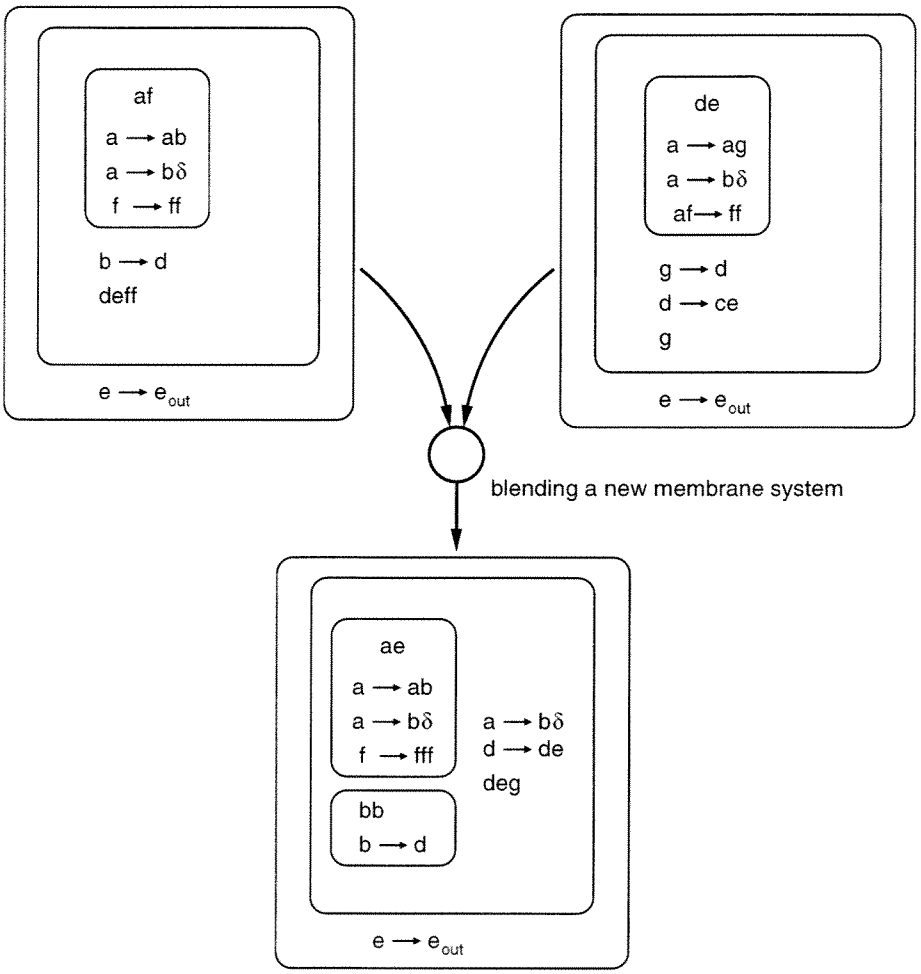

Figure 5.15: Illustration of one of the main goals of computational C-Blending: the creation of a new cell which contains new structure but also elements from two existing cells. In this thesis, blending between entire membrane systems is limited to blending between the contents of two single membrane systems only.

5.6.3 Basic Ideas and Principles: An Example

As explained in Section 5.4, blending is based on several constructing principles (Section 5.4.1) and guided by optimality and governing principles (Section 5.4.2).

In order to illustrate the basic ideas of C-Blending, let us consider the following simple membrane system, which simply transforms incoming molecules into a different outgoing molecule. The system consists of one (skin) membrane only.

Definition 5.6.1 (C-Blending Molecules)
A C-Blending molecule is a symbol from the alphabet $V = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$ *of 26 letters. The following multisets are valid:*

$$s_1 = a^2bijg^2h^2y$$
$$s_2 = l^2aksu^3$$
$$s_3 = a$$
$$s_4 = u^3$$

■

Note that Paun rather talks about *objects* instead of molecules. I prefer the term molecules, which shall be used throughout this the present work.

Definition 5.6.2 (C-Blending Reactions)
An C-Blending reaction is defined as

$$r = (u, s, v) = (u \longrightarrow sv) \tag{5.2}$$

where u, v *are multisets over the symbols of the alphabet* V *and* s *is a special symbol from* $S = \{\mathsf{H}, \mathsf{L}\}$. *When the* H *(HERE) as a special symbol is used, the reactions are normally applied. On the other hand, the* L *(LEAVE) symbol forces to move the resulting molecule outside the cellular membrane in the upper-next compartment or outside the skin membrane. Molecules outside the skin membrane are considered as an "output" of the system.*

Furthermore, let R *be the multiset of reactions* r. ■

The following table shows some valid reactions and their syntax in MATLAB, which is a little different from the usual syntax, but all the same pretty intuitive. The arrow is replace by a single dash "-", which is directly followed by the special symbol. The elements of each multiset (input an output) are separated by a vertical bar "|".

Reaction	MATLAB syntax
$a \rightarrow \mathsf{H}b$	a-H b
$ab \rightarrow \mathsf{H}a^2b$	a|b-H a|a|b
$c^2 \rightarrow \mathsf{L}abc$	c|c-L a|b|c

The following formalism describes the membrane system:

$$
\begin{aligned}
&\Pi = (V, T, C, S, \mu, w_1, R_1),\\
&V = T = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\},\\
&C = \emptyset,\\
&S = \{\mathsf{H}, \mathsf{L}\}\\
&\mu = [_1]_1,\\
&w_1 = \{\text{a multiset of objects}\},\\
&R_1 = \{\text{a set of rules:}(u_i \rightarrow z_i v_i)\}.
\end{aligned}
$$

The above-mentioned membrane system allows cooperative rules and therefore is, as already mentioned in Section 2.11.1, universal as long as all the rules are applied in a maximum parallel and synchronous manner.

5.6.4 Similarities and Activities

The C-Blending approach heavily relies on a similarity notion between strings and rules. As with blending, the goal is to be able to identify similar or opposite objects to from matching and counterpart connections. Ramscar and Yarlett stated: "In existing models symbol similarities are externally defined but neither empirically grounded nor theoretically justified" [331]. This is exactly the case for C-Blending, but is not a fundamental problem as our symbols (i.e., chemicals) do not represent knowledge and concepts which should be grounded.

Interestingly, a recent work conducted at the University of California San Diego investigates the relation of routine activity and conceptual integration and claims that they mutually influence each other [360].

> On the one hand, the intensity and extent of interaction with and coordination between artifacts finds a reflection in the extent of integration between the two domains. Activity gives rise to blends. On the other, the integration of domains may lead to action that would not have been performed in either of the original domains. Blending gives rise to activity. [360]

The activity Beate Schwittenberg was looking at is of course not directly comparable to the activity of chemicals as it will be defined below, nevertheless, the basic ideas share some similarities.

In order to introduce similarities between strings, let us take a look at the following multisets of objects (from the alphabet V as specified above):

$$s_1 = aab$$
$$s_2 = xzu$$
$$s_3 = ab$$
$$s_4 = zux$$

Remember that the order of the symbols does not matter in a multiset of objects. Intuitively, s_1 is similar to s_3, s_2 is similar to s_4, and s_1 opposite of s_2. This "intuitive" judgment makes use of the distance between the ordered alphabet and also between the difference in length.

We decided to use the following similarity definition, which is basically based on the ASCII codes of each letter. The letter "a" is associated with the integer 97, "b" with 98, ..., "z" with 120.

Definition 5.6.3 (String Similarity Measure)
Let $S_A = a_1 s_2 \ldots a_n$ and $S_B = b_1 s_2 \ldots b_m$ be two strings of symbols from the alphabet V. Each letter s is transformed in its ASCII code by means of the mapping function $ASCII(s)$.

$$d = |\sum_{i}^{n} ASCII(a_i) - \sum_{i}^{m} ASCII(b_i)|$$
$$max_l = max(n, m)$$
$$min_l = min(n, m)$$
$$d_{max} = (max_l * 120) - (min_l * 97)$$
$$similarity = \frac{d_{max} - d}{d_{max}}$$

■

The following MATLAB function allows to measure the similarity between two strings:

MATLAB AMB toolbox example:

```
>> s = stringSimilarity('ab','ab')
>> s = 1
>> s = stringSimilarity('a','z')
>> s = 0
>> s = stringSimilarity('aab','xzu')
>> s = 0.1067
>> s = stringSimilarity('house', 'garden')
>> s = 0.6883
```

The last example does of course not make much sense as humans have a their own concept of "house" and "garden". The above-defined similarity makes of course no sense as it works on a *purely* syntaxic level only!

The similarity between rules is based in the `stringSimilarity` function:

MATLAB AMB toolbox example:

```
>> s = ruleSimilarity('ab|sd-Lf','g-Ha')
>> s = 0.6276
>> s = ruleSimilarity('aa|nd-Lf','s-Hai|a')
>> s = 0.4323
```

The proposed measure is somehow arbitrary and one might well chose another measure based on further and/or other criteria. The only goal of this measure is to be able to compare objects and rules within the membrane systems.

In addition, each rule within a membrane system possesses an associated *activity* value, which, as its name says, indicates how active this rules has been in the past, i.e., how often it has participated in a reaction. The activity measure is defined as illustrated in the following equation:

$$activity = activity + \frac{1 - activity}{2}$$

The initial activity is set to 0, and the value the asymptotically towards 1. A rule which has been applied four times would have an activity value of $0 + 0.5 + 0.25 + 0.125 = 0.875$. The advantage of this measure is that it lies within the interval $[0, 1]$.

Both similarity and activity are then used to create the mapping between two membrane system's elements. Later, we'll in addition introduce a "fitness" value, which will depend on the performance of the membrane system with regards to a task it has to complete.

5.6.5 Blend Two Single Membrane Cells

Remember that the principal goal of C-Blending is to generate a new cell from two existing cells. This could of course also be done randomly, by simply choosing some elements and putting them together in a new cell, however, this basically already what evolutionary algorithms do with the crossover and mutation operators. In that sense, blending also an attempt to create new individuals in a more intelligent way.

Algorithm 10 shows the three main steps we adopted to blend a new cell. It is important to note that the C-Blending concentrates on the evolution rules and *not* on the molecules. For most applications, the molecules of an artificial chemistry might be considered as data whereas the rules might be

seen as a sort of program. This is the reason why the blending operations only work with the rules.

Algorithm 10 C-Blending: Blend Two Single Menbrane Cells

1: Create and initial mapping between the rules of each cell.
2: Improve the initial pairings.
3: Meld the pairings together to form a new cell.

Let Π_1 and Π_2 be two single membrane systems:

$$\Pi_1 = (V, T, C, S, \mu, w_1, R_1),$$
$$V = T = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\},$$
$$C = \emptyset,$$
$$S = \{\mathsf{H}, \mathsf{L}\}$$
$$\mu = [_1]_1,$$
$$w_1 = \{a, b, b, h, k, i, i, i\},$$
$$R_1 = \{a \to \mathsf{H}b, ab \to \mathsf{L}g, i \to \mathsf{H}dh, k \to k\}.$$

$$\Pi_2 = (V, T, C, S, \mu, w_2, R_2),$$
$$V = T = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\},$$
$$C = \emptyset,$$
$$S = \{\mathsf{H}, \mathsf{L}\}$$
$$\mu = [_1]_1,$$
$$w_2 = \{b, b, b, c, o, q\},$$
$$R_2 = \{a^2 \to \mathsf{H}b, b \to \mathsf{L}h, c \to \mathsf{H}c^2\}.$$

They are created in MATLAB as following:

MATLAB AMB toolbox example:

```
>> cell1 = initCell({'a','b','b','h','k','i','i','i'},...
   {'a-Hb','a|b-Lg','i-Hd|h','k-Lk'})
>> cell2 = initCell({'b','b','b','c','o','q'},...
   {'a|a-Hb','b-Lh','c-Hc|c'})
```

In order to update the membrane system's activities, let us first run for some — say twenty (this is a good value for this small chemistry) — steps both cells.

MATLAB AMB toolbox example:

```
>> cell1 = runCell(cell1,20);
>> cell2 = runCell(cell2,20);
```

After that, the molecules of each cell are as follows:

$$w_1 = \{b, d, d, d, h, h, h, h\} w_2 = \{b, c, c, o, q\}$$

The activities of each rule are illustrated in Table 5.2.

Cell	Rule	Activity
1	a-Hb	0
	a|b-Lg	0.5
	i-Hd|h	0.875
	k-Lk	0.5
2	a|a-Hb	0
	b-Lh	0.75
	c-Hc|c	0.5

Table 5.2: Rules and activities of Π_1 and Π_2.

As shown in Algorithm 10, C-Blending consists in creating in a first step a mapping between the two membrane systems Π_1 and Π_2. The initial mapping is basically done randomly and will in a second step be improved by exchanging existing pairings between rules. b-Lb' 'c-Hd|c' 'a|a-Lb' 'k-Lk In the case of our two cells, the initial mapping might for example look like illustrated in Figure 5.16. This is done in MATLAB with the function `newcell = initialPairings(cell1, cell2)`. This initial mapping is very natural to implement in an artificial chemistry and corresponds to the rules floating around in a reactor and sticking together when they meet.

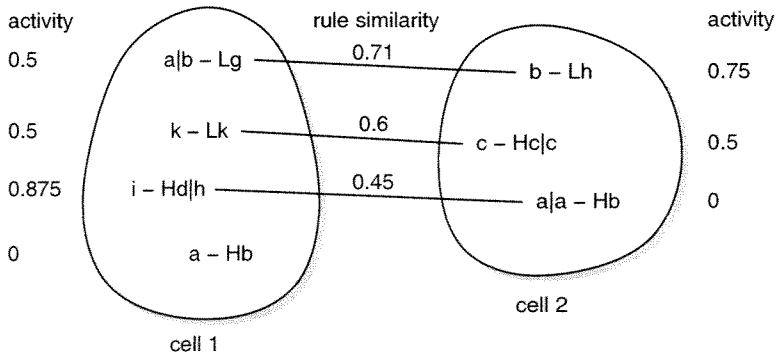

Figure 5.16: A possible initial mapping between the two membrane systems Π_1 and Π_2. The values on the connections between the cells indicate the combined similarity and activity of the according rules.

In the next step, the initial mapping will be improved by the function `newcell = improvePairings(newcell,steps)`, where `steps` indicates the

number of times a new pairing should be tried. Again, this is best compared as reactions floating around in a reactor and meeting. If an existing coupled pair meets another pair, they evaluate whether exchanging their partner would be of any advantage. The same happens when an existing coupled pair meets a single reaction. If a new pairing is of any advantage, one of the partners will have to exchange with the single reaction. From a global point of view, the goal is to make as many good mappings as possible, i.e., as may pairs which have similarity and activity values as close as possible.

Figure 5.17 shows an improved mapping. Intuitively, the mapping favors mappings between rules that are similar and have similar activities or that are opposite. This exactly what Faconnier and Turner's blending does. It can be seen in Figure 5.17 that the similarities are higher and that the activities are either similar or opposite.

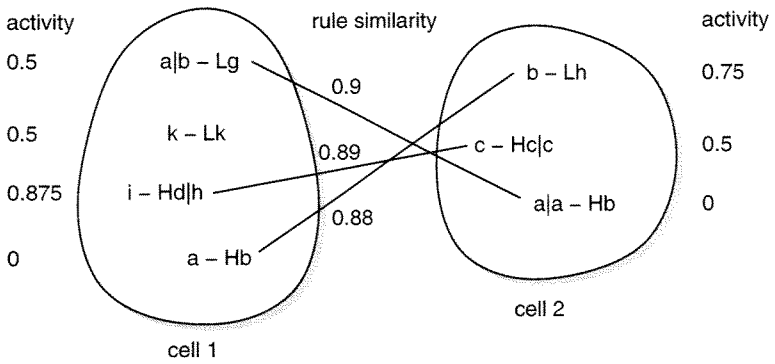

Figure 5.17: An improved mapping between the two membrane systems Π_1 and Π_2.

Finally, the improved pairings between the two membranes are used to blend the new cell (`newcell = meldPairings(newcell)` in MATLAB). This is simply done by composing a new rule for each mapping by randomly choosing elements from both rules that participate in the mapping. For example, from the rules `a|b-Lg` and `a|a-Hb`, the following rule has been blended: `a|a-Lb`. This last step is somehow similar to the crossover operator in evolutionary computation.

Note that two cells can easily be "blended" together in MATLAB to a new cell by the function `newcell = blendCells(cell1, cell2)`.

As it should be, C-Blending is nondeterministic and blending two cells together for two times does not result in the same cell, as the following example shows: the reactions of the blended cell `b1` and `b2` are not the same.

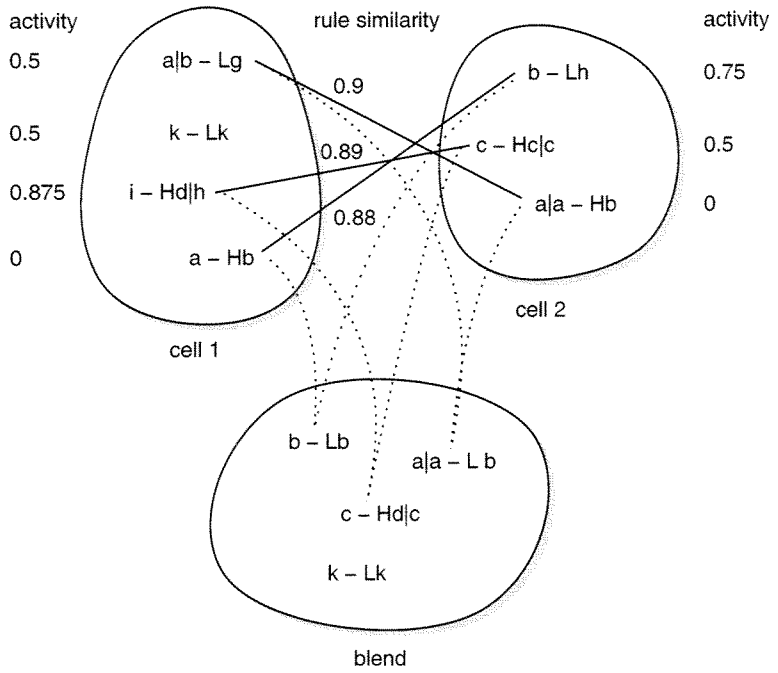

Figure 5.18: The blended cell.

MATLAB AMB toolbox example:

```
>> c1 = initCell({'a','b','c'}, {'a-Hb|c','a-Ha'});
>> c2 = initCell({'a','b','c'}, {'a|b-Hc','a-Lc|a'});
>> c1 = runCell(c1,10);
>> c2 = runCell(c2,10);
>> b1 = blendCells(c1, c2);
>> b2 = blendCells(c1, c2);
>> b1.reactions
>> 'a|a-Ha'    'a-Ha|a'
>> b2.reactions
>> 'a-Ha'    'a-La|c'
```

Recall that the main goal of the Divago computational blending system (Section 5.5.4) was to model divergent thought in problem solving by proposing solutions outside the domain focus. This is exactly what C-Blending does: it proposes a new chemistry which is partially based on the two existing chemistries, but also contains new reactions.

Finally, it is worth mentioning that the proposed method can more or less straightforwardly be implemented in a completely decentralized system which is based on a chemical signals only. We shall see the principles of such an implementation later.

5.6.6 Blending in a Population of Membranes

From the point of view of genetic algorithms, each membrane system might be considered as an individual in a population. Let us consider a set of membrane systems where each membrane has to perform the same task. Algorithm 11 illustrates the method we successfully tested (see Examples in Section 5.6.9)

Algorithm 11 Blending Cells in a Population

1: $t = 0$.
2: Initialize a population $P_N(t = 0)$ of N cells.
3: Evaluate population $P_N(t = 0)$.
4: **while** not finished **do**
5: $t = t + 1$.
6: Create an initial 1-by-1 random mapping between a subset of B cells within the population (Figure 5.19).
7: Improve the mapping based on the "cell's attractivity" (Figure 5.20).
8: Blend the B 1-by-1 mappings to form B new cells (Figure 5.21)
9: Remove B cells with the lowest fitness from the population $P_N(t)$ (Figure 5.22).
10: Evaluate population $P_N(t)$.
11: **end while**

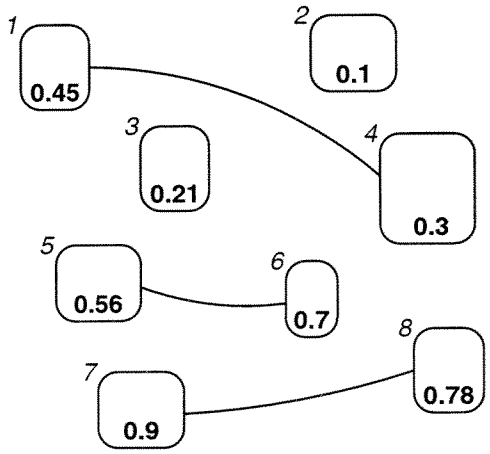

Figure 5.19: An initial random mapping in a population of membranes. The numbers within the cells indicate the cell's attractivity.

Algorithm 11 holds the population size constant through the removal of the same number of bad cells as new cells have been added. Of course, other algorithms are imaginable: for example, one could only blend new cells

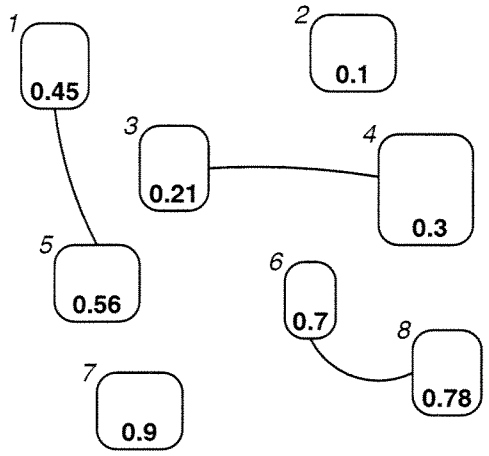

Figure 5.20: An improved mapping in a population of membranes. The difference between the cell's attractivity has been minimized.

from a mapping between the best cells and only remove cells when a certain population size has been reached. One could also imagine a replacement algorithm where the oldest individuals die out. All these issue have been extensively addressed in the field of evolutionary algorithms (see for example [120, 407]) already. The key of C-Blending is not the replacement algorithm in a population but the way new individuals are being created from existing ones.

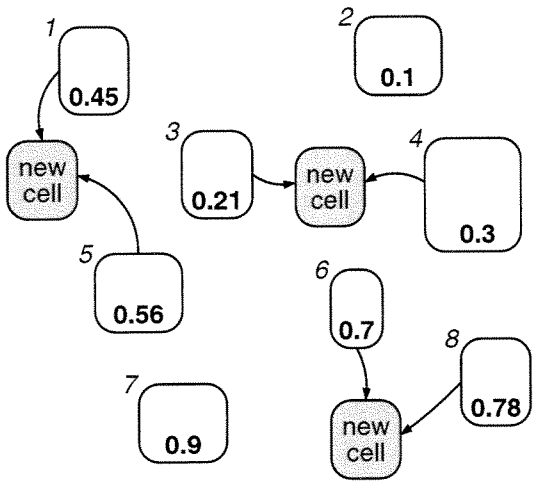

Figure 5.21: A new cell is blended from each 1-by-1 mapping.

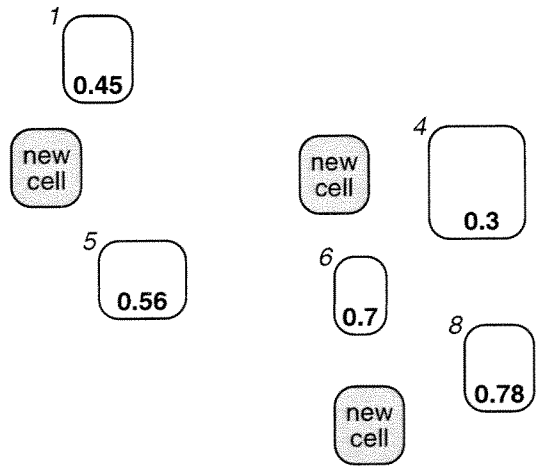

Figure 5.22: In order to hold the population size constant, cells with the lowest fitness values will be removed from the population.

5.6.7 Fitness, Feedback, and Optimality Principles

Evolutionary algorithms and in general also neural networks can only work when a clearly specified goal is available. They work best when for each possible input its desired output is known. If that is the case, a precise feedback can be given and the system's "only" goal is to minimize the global error.

In living systems, it is in most cases impossible to directly specify an output for a given configuration of the inputs. One reason — among many others — is that often inputs and outputs cannot even been clearly identified. Furthermore, the environment very rarely gives an indication on how and in which direction the error should be corrected. Most often, we just know that something was wrong and that we have to try it in a different way. This idea has been used for the classical reinforcement learning method (for a good introduction see for example [397]). An individual only receives a *reinforcement* or a *punishment* (might be a negative reinforcement) depending on how well it performs on a given task. No further quantitative indication is provided on the error.

As presented above, the fitness value of each cell within a population is used to find appropriate pairings between cells in order to blend new cells. The idea and hope behind this is of course that the chance to obtain a good cell from two already well performing cells is higher than if one would randomly choose cells.

Whereas a fitness-based selection is appropriate for most optimization problems, one could image various other measures. In the domain of concepts, the similarity or proximity of two different concepts might for example be used to form pairings.

As detailed in Section 5.4.2, the governing and optimality principles play a crucial role for blending. The principles are used to form "good" blends and characterize strategies for optimizing emergent structure. They frequently compete with each other and that results in a variety of lower-level optimality pressures for constructing the blend, so one should not expect to observe them in any given integration in a perfect way, according to Fauconnier and Turner [130].

In [324], Pereira and Cardoso present a formalization of the optimality principles for their own computational blending approach. By means of a genetic algorithm they try to find good blends that respect the optimality principles.

My own point of view as well as the application domain are somehow different: as I use blending as a kind of optimization technique to find cells that perform a certain task, the natural-selection-based algorithm ensures that only the best cells survive. In addition, the activity-based mapping of the reactions allows to cover several optimality principles on its own. I

therefore claim — and the experiments performed confirm this — that C-Blending does not require any optimality constraints as it is based on a natural selection.

However, some further considerations shall be given regarding the chemical based blending:

- **Compression** might be accelerated and improved by a special special compression algorithm which simplifies reactions. For example, two reactions of the form $a \rightarrow b$ and $b \rightarrow c$ might be reduced to $a \rightarrow c$.
- The **topology principle** is ensured by the activity-based mapping between the reactions. The closer the activity of two reactions, the more likely it is to map them together.
- **Unpacking** is usually not possible in an unambiguous way as the reactions are not reversible.
- **Relevance**, **vital relations**, **integration** are basically guaranteed by the activity and the fitness. Active reactions are considered more relevant and important.

5.6.8 Limited Cellular Resources

Nature is basically finite and it makes little sense to build hypothetical bio-inspired machines which allow to store an infinite amount of information.

I order to make our cells a little more realistic though, I decided to limit their resources from the beginning on. A cell starts loosing reactions and molecules when the maximal capacity has been overstepped. The maximal capacity can be defined for each cell separately (`cell.maxresources` in MATLAB).

This (so far virtual) limitation especially also makes sense in view of future hardware implementations, where one has to deal with limited resources. As we will see in the next chapter, a cell will be built up from a distributed network of chemical reactors with limited capacity for hosting reactions and molecules.

5.6.9 Examples and Applications

Example 5.6.1 (Transforming molecules I)
Let us consider two very simple toy applications. The goal is to obtain cell which are able to recognize molecules and to reply with another molecule (similar to pattern recognition). More precisely, the cell gets as an input five identical molecules from the alphabet $V_{in} = \{a, b, c\}$. *The cell must*

---

then be able to transform the five input molecules in at least three output molecules, namely: $a \rightarrow x$, $b \rightarrow y$, *and* $c \rightarrow z$.

In this particular case, the population of cells is updated and evaluated synchronously not unlike in a classical genetic algorithm. Algorithm 12 illustrates the proceeding. Each cell contains a two special "compartments", an inset *and an* outset *(Figure 5.23). Molecules in the inset will be transferred as soon as a cell is run into the main membrane, whereas molecules that leave the cell by means of the special operator L will end up in the outset. Both inset and outset have actually been introduced because of implementational issues: they provide a buffer between the other cells. The outset can also simply be considered as the "outside" of the membrane.*

Algorithm 12 Pattern recognition

```
1: Initialize all N cells random rules.
2: epochs = 0
3: while epochs < max_epochs do
4:    nbpres = 0 (number of pattern presentations)
5:    while nbpres < maxpres do
6:       Present a set of input molecules to each cell (i.e., add the molecules
         to the inset).
7:       Run each cell for s steps.
8:       Evaluate each cell (i.e., are the desired molecules present in the
         outset?) and accumulate the fitness.
9:       nbpres = nbpres + 1
10:   end while
11:   Build an initial two-by-two random mapping between the cells.
12:   Improve the mapping.
13:   Blend the cell pairs and create new cells.
14:   epochs = epochs + 1
15: end while
```

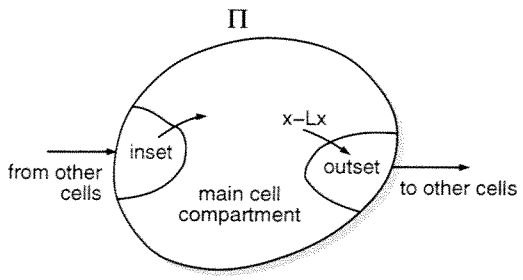

Figure 5.23: Illustration of the *inset* and *outset*.

Figure 5.24 shows the evolution of the performance of the population of $N = 20$ cells. The system was run for 100 epochs and each cell was run for 20 steps to apply its rules to the molecules. The multisets of molecules presented at the inputs were $I_1 = \{a, a, a, a, a\}$, $I_2 = \{b, b, b, b, b\}$, and $I_2 = \{c, c, c, c, c\}$. The cell obtains a maximum fitness value if it is able to transform each of the three multisets into a multiset of at least three instances of x, y, or z. In addition, Figure 5.25 shows the resources used. During each epoch, only three new cells were blended (the three worst-performing cells were removed from the population) and the maximum number of resources (molecules and reactions together) was limited to 20. The initially and randomly generated reactions were of the form $s \rightarrow Ls$ or $s \rightarrow \mathsf{H}s$ only, where x is a single symbol from the alphabet $V = \{a, b, c, x, y, z\}$.

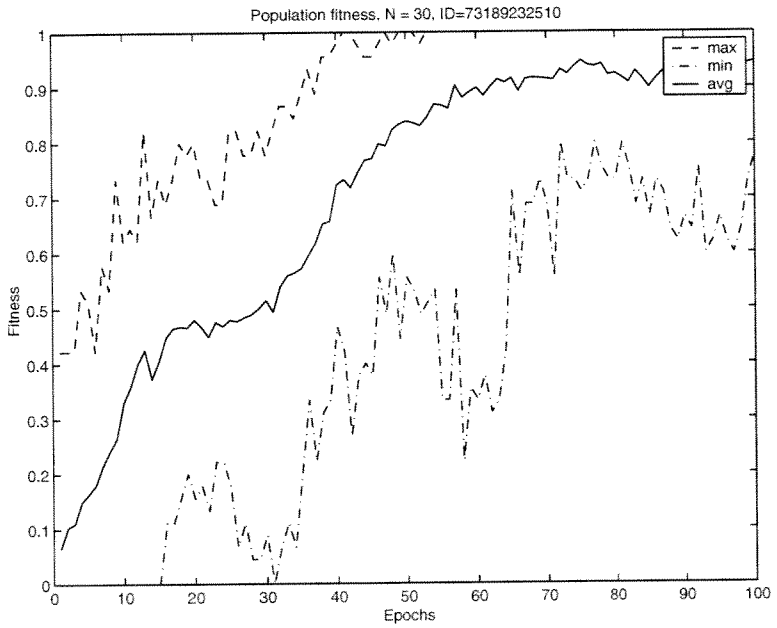

Figure 5.24: Fitness graph of a simple pattern recognition experiment.

One of the cells which was able to solve the task contained the following set of reactions:

```
a-Lx
a-Lx
a-Lx
a-Lx
a-Lx
b-Ly
b-Ly
```

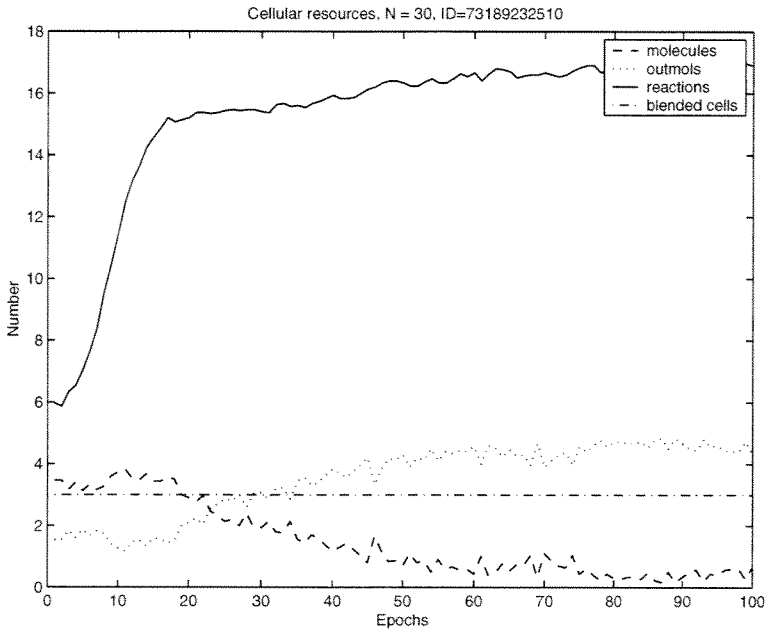

Figure 5.25: Resources used in a simple pattern recognition experiment.

```
b-Ly
b-Ly
c-Lz
c-Lz
c-Lz
c-Lz
c-Lz
c-Lz
c-Lz
```

As one can see, there are several copies of each reactions which allow to transform the incoming molecules. ■

Example 5.6.2 (Transforming molecules II)
The second example is a little more complicated as there are rules of the following forms are allowed:

$$
\begin{aligned}
s_1 &\rightarrow S s_2 \\
s_1|s_2 &\rightarrow S s_3 \\
s_1 &\rightarrow S s_2|s_3 \\
s_1|s_2 &\rightarrow S s_3|s_4
\end{aligned}
$$

Such rules can be easily generated by the following MATLAB *command:*

MATLAB AMB *toolbox example:*

```
>> randomReactions({'a','b','c','x','y','z'}, {'L','H'},...
                   [1 10], [1 1], [1 1], [1 2], [1 2]);
>> 'c|y-La|a'    'z-Lz|c'    'x-Lc|b'    'x|a-Lc'
```

The symbol s_i is randomly chosen from the alphabet $V = \{a, b, c, x, y, z\}$, $S \in \{\mathsf{L}, \mathsf{H}\}$. Figures 5.26 and 5.27 show the results. The population size was again $N = 30$ cells, but six cells were blended at each epoch this time.

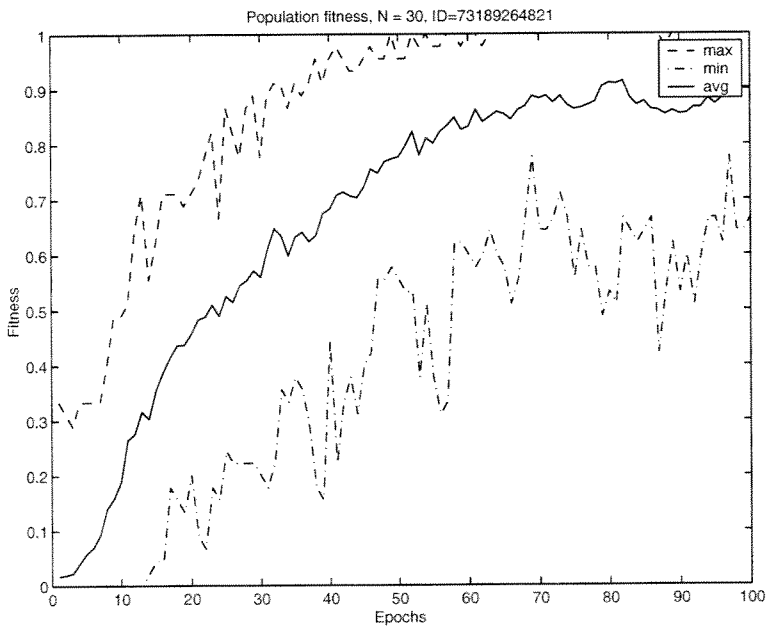

Figure 5.26: Fitness graph of a simple pattern recognition experiment with more complicate rules.

■

Example 5.6.3 (Transforming molecules III)
In this last example, the symbol alphabet has been extended to $V = \{a, b, c, d, e, f, u, v, w, x, y, z\}$, which enlarges the search space six times. Figure 5.28 shows the fitness plot and one can see around epoch 250, one succeeding individual was found. The population size was $N = 40$ cells,

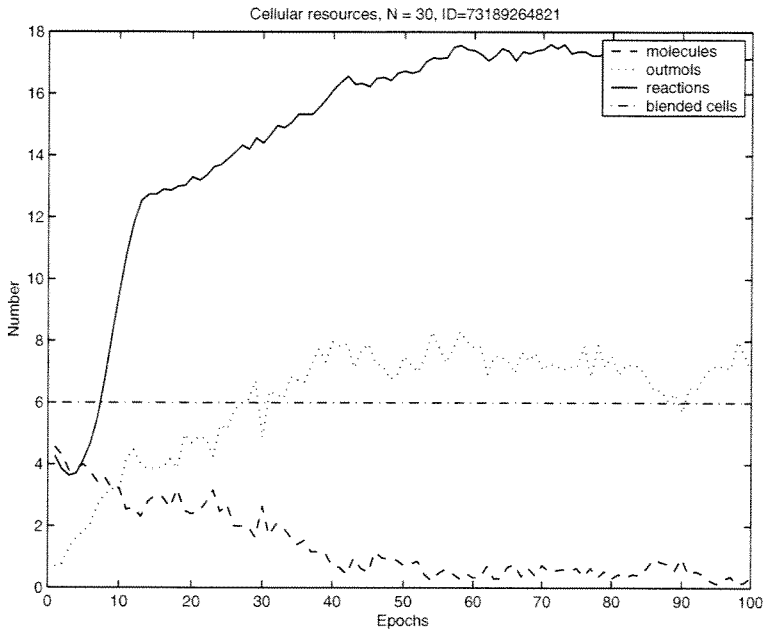

Figure 5.27: Resources used in a simple pattern recognition experiment with more complicate rules.

the maximum number of molecules and reactions per cell was limited to 50.

■

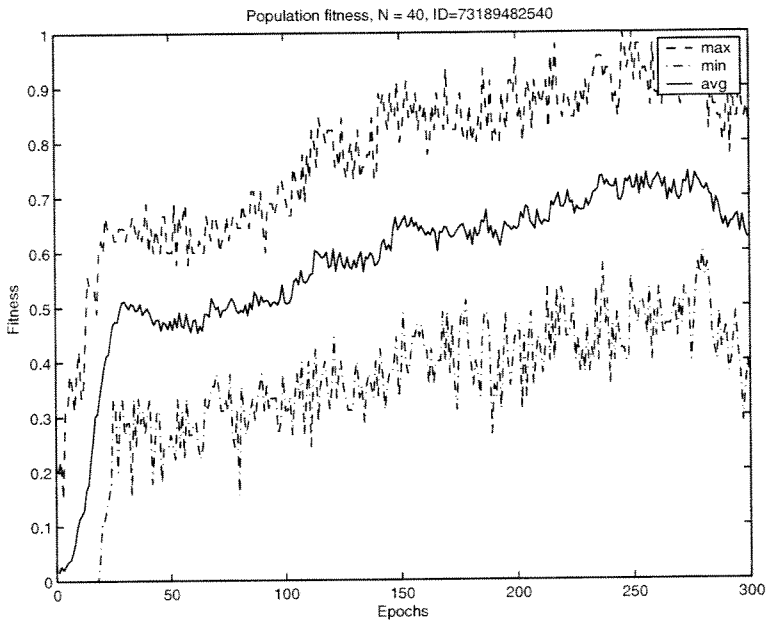

Figure 5.28: Fitness graph of a simple pattern recognition experiment with an extended alphabet.

5.6.10 Blending Hierarchical Cell Structures

So far, we have only considered blending between two single membrane cells and their contents. The vision of blending entire hierarchical membrane structures as illustrated in Figure 5.15 will not be addressed in detail in this thesis.

Figure 5.29 however delineates the main idea of blending between two membrane systems: the method of blending between single cells will be applied between similar cells at similar levels in the membrane structure. The question remains open — and shall not be addressed here — what to do with totally different membrane structures, i.e., blending between a membrane system with five and one with 25 membranes.

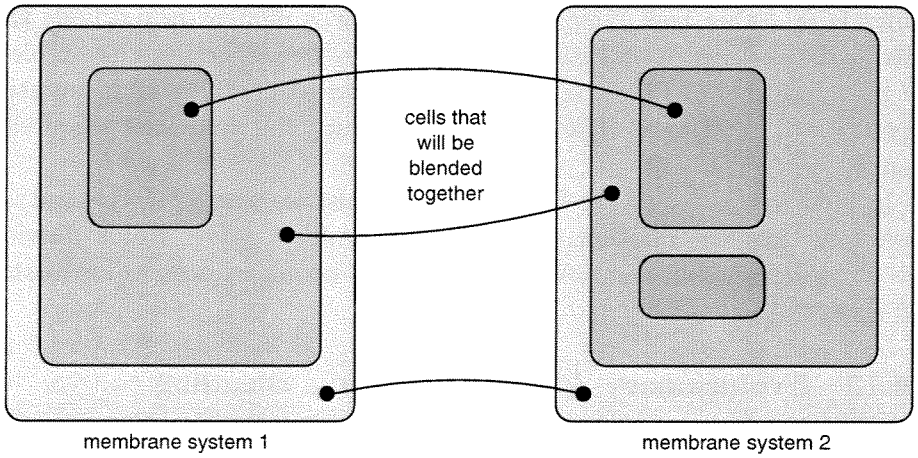

Figure 5.29: Blending between to hierarchical membrane systems consist in applying blending to similar cells within each membrane system.

5.6.11 A Guide to Blending in Artificial Chemistries

The C-Blending method presented so far basically represents one possible method for blending between a certain class of membrane systems. A lot of other methods and possibilities can be imagined, which might perform better for certain applications are which might be more suitable for other classes of P systems.

The following list shall provide a sort of a "recipe" to what has to be decided when "inventing" other methods of computational blending.

- Define the objects (i.e., the molecules and reactions) to be used. Decide whether blending shall use molecules or reactions only or both together.

- Define the membrane structure and the reaction dynamics.
- Define a measure which allows to compare (i.e., find matching and counterpart connections) the objects (i.e., the molecules and/or reactions) within a cell.
- Define a method which will allow to combine matching objects.
- Define a fitness measure to evaluate the performance of each cell.
- Define a measure which allows to compare the cells within a population of cells. This will allow to identify cellular parings used to blend new cells.

As I said above, a lot of other methods might be imagined. This difficulty is probably best compared with the large number of methods that exist in the domain of evolutionary algorithms. Depending on the problem, the encoding, the population size, etc., a different crossover and mutation operator will be chosen. The framework remains the same, but the details change.

In blending, there might be artificial chemistries where one method will work well, but not another. The general problem solver which provides optimal methods and solutions to all kinds of problems does unfortunately not exist.

5.6.12 Evolutionary Algorithms versus Blending

Evolutionary algorithms have already been introduced in Section 2.3 (see also [297,407] for an introduction). In this section, I'd like to point out some similarities and contrasts between blending and evolutionary algorithms.

Let's first start with an interesting point mentioned by Pereira and Cardoso [321]: they consider a genetic algorithm itself as an example of a blend, although they use it to find blends that fit the optimality principles (Section 5.5.4).

I cannot fully agree with this point of view (i.e., to consider genetic algorithms as blends), however. My main objections are based on the following points:

- The operators of genetic algorithms (i.e., crossover and mutation) work on the *genotype* of an individual within a population. If one considers a mental space as an individual (within the population of mental spaces), then the operators of blending work on the *phenotype* instead of the genotype.
- Each individual within a population usually has to complete the same task. It is therefore not appropriate to consider mental spaces as members of a population.

- In genetic algorithms, an externally evaluated fitness value for each individual is required. In blending, the internally defined optimality principles are used to blend "good" blends.

- Genetic algorithms and blending of course work — at least in Nature — on completely different timescales. Cognitive processes can be very fast and often work in the blink of an eye.

Nevertheless, there is one rather obvious similarity: evolutionary biological processes do also produce something new (i.e., new species and individuals), but take place over extremely long time spans. For more details on the analogies with evolutionary biology see Turner [427, p. 142ff].

In summary: from all this I would definitely say that genetic algorithms and evolutionary biological processes should not be considered as an example for blending.

5.7 Wrap-Up

In this chapter, I have first presented Fauconnier and Turner's conceptual integration (or blending) framework which provides a theory how humans create new concepts based on existing knowledge. The most exciting part of this theory is probably the emergent structure a blend usually creates. The most obvious drawback of the theory of blending is that it is not at all a formal theory and that a direct link to the underlying neural levels is yet to be confirmed.

In the second part of this chapter, I presented the C-Blending method inspired by blending to create new membrane systems based on already existing membrane systems. The main goal was to use the basic mechanisms of blending and to apply them to membrane systems and artificial chemistries in the form of a new computational blending approach. As we have seen, C-Blending did only deal with the cell's reactions but *not* with the molecules. One could however easily imagine the same approach applied to molecules too. The same similarity measure might be used and the activity might also be defined in the same way. The reason I didn't do this was basically motivated by the insight that the rules play a more important role in a membrane system than the molecules. The molecules might be considered as data, whereas the multiset of reactions forms a sort of program.

Interestingly, the presented C-Blending approach has only a few similarities with evolutionary algorithms (see also previous section), but is actually much more similar to *classifier systems.*

According to [407], a classifier system has three main components:

- a message and rule system;

- a credit-assignment system; and
- a genetic algorithm.

Basically, a classifier system as illustrated in Figure 5.30 tries to match messages with of or more classifier conditions of the form:

$$\text{IF } \langle condition \rangle \text{ THEN } \langle action \rangle$$

Each classifier gets a fitness value through a credit-assignment system and new rules are discovered by means of a genetic algorithm. The new and possibly superior rules are obtained by classical genetic operators (recombination and mutation). The main difference to classical genetical algorithms is that the entire population of classifiers co-evolves and collectively learns.

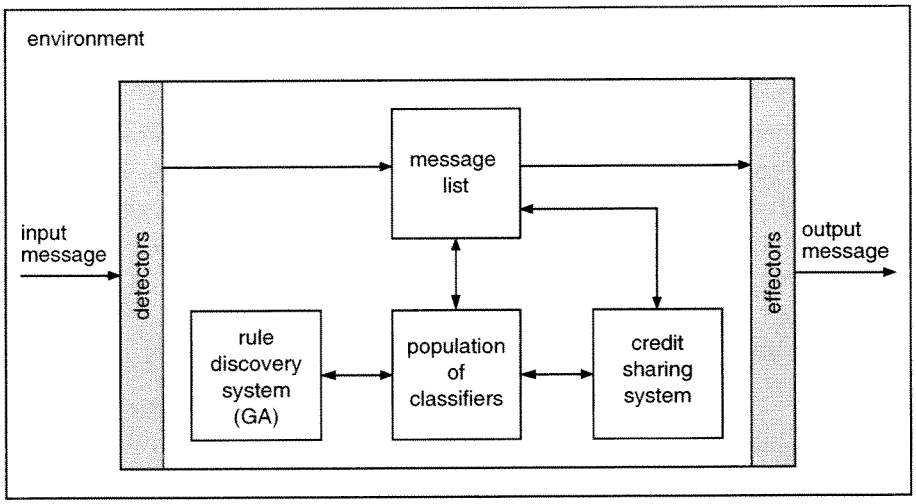

Figure 5.30: Architecture of a learning classifier system. Redrawn from [407].

The C-Blending method shares several elements and characteristics of classifier systems:

- The classifier conditions are the counterparts to the chemical reactions.
- The population of classifier is equivalent to the set of all reactions.
- The credit sharing system might be compared to the activity- and similarity-based approach of C-Blending.
- Finally, the rule discovery system is the counterpart to the blending-based approach coupled together with a natural selection algorithm.

The basic concepts of the proposed C-Blending approach have been tested (proof of concept) by means of very simple toy applications. For larger tests, a new simulator — written for example in C or C++ would be necessary in order to overcome computational limitations.

CHAPTER 6

Towards Membrane Blending: atP, aP, and aB Membrane Systems

> It is easier to tone down a wild idea than to think up a new one.
>
> Alex Osborne (the father of brainstorming)

6.1 Introduction

Hierachically organized cellular structures — such as membrane systems — represent a powerful computational tool which might fully exploit the parallelism of an eventual underlying, massively parallel hardware. This is exactly what an amorphous computer offers: massive and fine-grained parallelism. Amorphous computing (see also Section 2.9 for a comprehensive introduction) is a promising approach which intends to develop organizational principles and programming languages for obtaining coherent global behavior from the cooperation of myriads of unreliable, locally interconnected parts that are interconnected in unknown, irregular, and time-varying ways.

Unifying membrane systems and amorphous computers would offer several interesting characteristics. One of the main reasons is that such an implementation might fully make use of the massive parallel and fine-grained architecture of the amorphous computer and the membrane system. As we have already seen in Chapter 4, P systems were usually simulated on a

single-processor machine and their inherent parallelism could therefore not be fully exploited. Paun writes, "[...] As long as we do not have genuinely parallel hardware on which the parallelism [...] of membrane systems could be realized, what we obtain cannot be more than simulations, thus losing the main, good features of membrane systems" [315, p. 379].

The main goals of this chapter are

- to propose classes of membrane systems in view of a realization on an amorphous computer and
- to unify the computational C-Blending approach with the membrane systems proposed.

Note that the implementation of the membrane systems on an amorphous computer will be presented in Chapter 7 and that the present chapter only deals with the cellular level of the membrane systems as well as with the the implementation of C-Blending on the membrane systems.

Figure 6.1 summarizes the two main levels we are dealing with in this chapter: (1) the Circuit Amorphous Computer on the lowest level and (2) aP membrane systems built upon the amorphon computer.

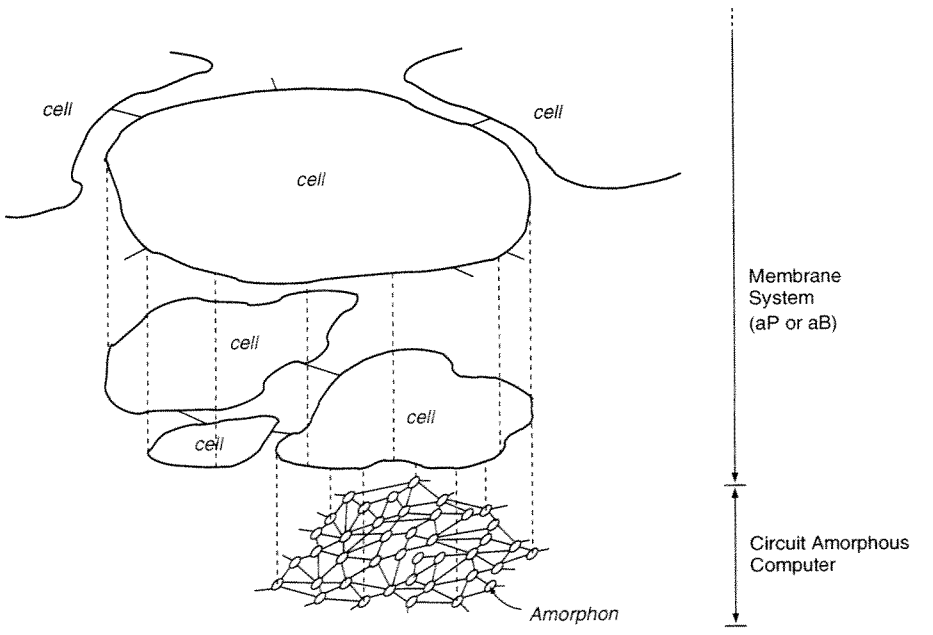

Figure 6.1: Illustration of the Circuit Amorphous Computer and the membrane system levels. In this chapter, we will only deal with the cellular levels. The Circuit Amorphous Computer will be presented in Chapter 7.

For an effective and more or less straightforward implementation of P systems on an amorphous computer, the following points seem to be important:

- The artificial chemistry's reactor dynamics should allow to separate and distribute the functionality on several reactors, each reactor having limited resources only.
- The chemistry must be tolerant to a certain loss of information, i.e., lost molecules and reactions, faulty membranes, faulty interconnections, etc.
- No global clock and other global signals should be assumed.

One of the main goals was to provide a basis for novel computational architectures based on what might be called "chemical metaphor". The central points are given below and the reaction of two molecules is illustrated in Figure 6.2.

Proposition 6.1.1 (Chemical Metaphor)

- *The entire system is based on abstract* molecules *and* reactions *only. Molecules might be considered as data, reactions as part of a program.*
- *Reactions and molecules are usually present in multiple instances. One can therefore also talk about a certain* concentration *of molecules and reactions.*
- *All communications are ensured by means of* gradients *which spread throughout the system.*
- Cellular membranes *allow to form compartments and to build hierarchical and modular systems.*

■

From a programmer's point of view, the chemical-based system as illustrated in Figure 6.2 might be considered as a set of *if-then* instructions, which are all executed in parallel (or according to another execution policy). Each reaction is basically substituted by one if-then instruction. The molecules are then triggering these instruction, which in turn will remove the triggering-molecule and replace it by another.

In this chapter we will introduce several membrane systems that will used henceforth. The atP membrane system was a first candidate model, which is later further developed to the aP membrane system. In the last section of this chapter, the aB membrane system will be presented, which unifies C-Blending and membrane systems.

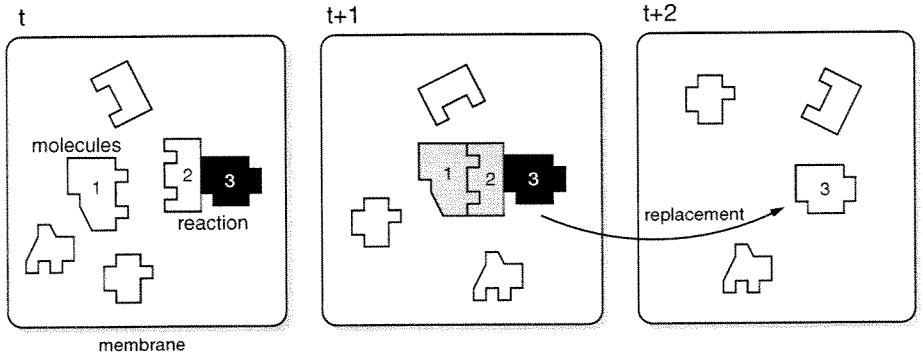

Figure 6.2: Illustration of the chemical metaphor. Molecule 1 matches the receptor 2 of the reaction and replaced by a molecule 3.

6.2 The atP Membrane System: A First Candidate Model

6.2.1 Tissue Membrane Systems

As a first example and candidate model for an implementation on an amorphous computer, I considered a membrane system inspired by *tissue* P *systems*. A *tissue* P *system* (tP system), introduced by Martín-Vide *et al.* [250], is a variant of P systems that processes symbols in a network of cells. Each cell has a finite state memory, processes multi-sets of symbol-impulses, and can send impulses ("excitations") to the neighboring cells. Martín-Vide *et al.* have shown that it is possible to simulate a Turing machine with a small number of such cells only. A cell might be considered as an abstraction of a biological cell or neuron, the interconnections as protein channels or synapses. The standard rewriting rules used are of the form $sM \rightarrow s'M'$, where s, s' are the cell's internal state and M, M' are multisets (intrinsically unordered) of symbols.

Formally speaking, a tP system [250] is a construct

$$\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, i_{out}), \tag{6.1}$$

where:

1. O is a finite non-empty alphabet (of *objects*);
2. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ (*synapses* among cells);
3. $i_{out} \in \{1, 2, \cdots\}$ indicates the *output cell*;
4. $\sigma_1, \ldots, \sigma_m$ are cells of the form

$$\sigma_i = (Q_i, s_{i,0}, w_{i,0}, P_i) \quad 1 \leq i \leq m, \tag{6.2}$$

where:

1. Q_i is a finite set (of *states*);
2. $s_{i,0} \in Q_i$ is the *initial state*;
3. $w_{i,0} \in O^*$ is the *initial multiset* of objects;
4. P_i is a finite set of *rules* of the form $sw \rightarrow s'xy_{go}z_{out}$, where $s, s' \in Q_i, w, x \in O^*, y_{go} \in (O \times \{go\})^*$ and $z_{out} \in (O \times \{out\})^*$, with the restriction that $z_{out} = \lambda$ for all $i \in \{1, 2, \ldots, m\}$ different from i_{out}.

The objects which appear in the left hand multiset w of a rule $sw \rightarrow s'w'$ are sometimes called *impulses*, while those from w' are also called *excitations*. The rules can be applied in three different ways (minimal, parallel, and maximal mode) and communication can be replicative, non-replicative, or non-deterministical. The combination of these possibilities gives us nine possible behaviors of a tP system (see [250] for more details).

Let us consider an very simple (toy) system, called atP, in order to illustrate the basics. As we will see, atP systems do not use internal states and the reaction dynamics will be different form the rule application policies of tP systems.

A atP system can be graphically represented as shown in Figure 6.3. Ovals are associated with cells and the arrows indicate the interconnections between the cells. A cell is bordered by a *cellular membrane*. Each cell can receive and send symbols to the environment. The interconnection topology is fixed and determined by the interconnection structure of the underlying amorphous computer.

At the beginning, each cell contains a set of initial symbols and initial rules. In the following, the details of the chemistry shall be described. Using the definition of Dittrich *et al.* [112], an artificial chemistry is a triple (S, R, A) where $S = s_1, s_2, \ldots, s_n$ is a set of molecules, a set R of reaction rules describing the interactions between the molecules, and an algorithm A that describes how the set R of rules is applied to a collection of molecules.

6.2.2 atP Molecules and Multisets

An atP molecule is a symbol from an alphabet V. The set of all molecules within a cell is a multiset M that can also be represented as a string $w \in V^*$. V^* denotes the language of all strings over V, including the empty string, denoted λ. $\Psi_V(w)$ gives the multiplicities in M of the symbols of V. Obviously, all permutations of w are representations of the same multiset.

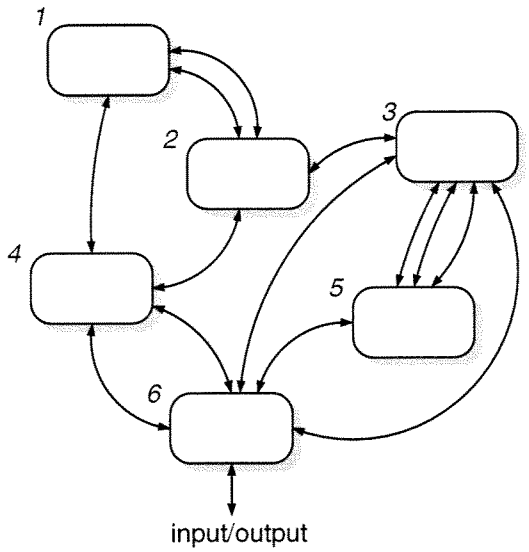

Figure 6.3: Graphical representation of an atP system.

Hence, the two multisets M_1 and M_2 (see also Figure 6.4) are identical ($V = \{a, b, c, d\}$):

$$M_1 = aabcadda \tag{6.3}$$

$$M_2 = ddbcaaaa \tag{6.4}$$

$$M_1 = M_2 = a^4bcd^2 \tag{6.5}$$

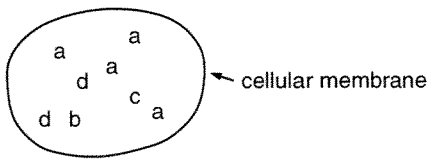

Figure 6.4: A multiset of an atP system.

6.2.3 atP Reactions

The *reactions* (also called *reaction rules* or simply *rules*) can be given *explicitely* or *implicitly*. Implicit reaction rule definitions are usually used for constructing chemistries. In a *constructive dynamical system* new components can appear that may change the system's dynamics. A system is *weakly constructive* [141] if the new components are created randomly and it is called *strongly constructive* when new components are generated through the interaction with other components (see also Section 2.10).

An *evolution rule* or *reaction* defines how the strands will be evolved. Reactions are usually written in the form of a chemical reaction. The following two notations are possible:

$$\begin{aligned} r = (u \longrightarrow v) &= (o_1 o_2 \ldots o_i \longrightarrow o'_1 o'_2 \ldots o'_j) \\ &= (u, v) = (o_1 o_2 \ldots o_i, o'_1 o'_2 \ldots o'_j) \end{aligned} \tag{6.6}$$

For a better readability, the reaction might also be written as follows:

$$\begin{aligned} r = (u \longrightarrow v) &= (o_1 | o_2 | \ldots | o_i \longrightarrow o'_1 | o'_2 | \ldots | o'_j) \\ &= (u, v) = (o_1 | o_2 | \ldots | o_i, o'_1 | o'_2 | \ldots | o'_j) \end{aligned} \tag{6.7}$$

The i molecules on the left hand side can react under certain conditions and will be subsequently replaced by the j molecules on the right hand side. The length of u, i.e, i, is called the *radius* of the rule. The chemistry is said to be *cooperative* if it contains rules of radius greater than one, otherwise it is said to be *noncooperative.*

In order to simplify a possible hardware implementation, we will restrict the length of u to $i = 2$. Hence, only reactions between up to two molecules are allowed. This is not a severe restriction as more complex reactions can be reduced to several two-molecule reactions with intermediate steps. Let us define the atP reactions as follows:

Definition 6.2.1 (atP Reaction)
An atP *reaction is defined as*

$$r = (u, s, v) = (u \rightarrow sv) \tag{6.8}$$

where u, v are multisets over the symbols of the alphabet V and s is a special symbol not in V.

Let R be the multiset of reactions r. ■

In contrast to normal P systems, the atP system allows to have several instances of the same reaction in the (multi-) set of reactions R. Furthermore, several special symbols are available that allow to implement special operations (see Table 6.1). Hence, the reaction $a \rightarrow \mathsf{L}b$ produces a b when the molecule a is present and sends b (in the same step) to all connected neighboring cells.

Symbol	Name	Description
H	HERE	Only apply the reaction, otherwise do nothing
L	LEAVE	Sends the resulting molecule to the neighboring membranes (all that are connected)

Table 6.1: Special symbols used in the atP system.

6.2.4 atP Reactor Dynamics

The *reactor dynamics* or *reactor algorithm* determines how the set of reactions is applied to the set of molecules. The reactor dynamics in basic P systems is such that the reactions are applied in a maximum parallel manner, i.e., all reactions that can be applied have to be applied. Furthermore, several P systems also deal with priorities: $(ff \rightarrow h) > (f \rightarrow g)$. In that case, the reaction $ff \rightarrow f$ has to be applied since it has a higher priority (see also [315, p. 70]). Assume that a region contains the multiset fff. In that case, the rule $ff \rightarrow h$ has to be applied and it is forbidden to apply rule $f \rightarrow g$) since it has lower priority. The resulting multiset is therefore: fh. In terms of biochemistry, rules with higher priority are more active. Note that there is a competition for rule application, but not for the objects to be used.

Priorities are copious to implement in a reactor that works in a distributed and decentralized manner. Reason why the atP chemistry does not make use of priorities. In our case, the molecules are represented in the form of a multiset without a particular spatial structure. A choice has to be made whether each molecule is treated explicitely or whether all molecules of the same type are represented by their frequency or concentration. A typical example is the well-stirred reactor (see Figure 6.5) that models stochastic reactions.

For our purposes, we will use Algorithm 13 to model stochastic collisions (in a closed reactor) between atP molecules and reaction rules. It is easy to see that the probability that a certain reaction or a certain molecule is selected is proportional to the number of instances in the according multiset.

6.2.5 atP System Formalization

Formally speaking, an atP membrane system is a construct

$$\mathsf{atP} = (V, Z, \sigma_1, \ldots, \sigma_m, C), \tag{6.9}$$

where:

1. V is an alphabet;

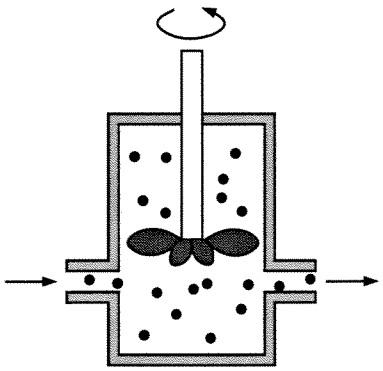

Figure 6.5: Schematic illustration of a well-stirred (open) reactor with input and output. Molecules leave the reactor with a certain probability.

Algorithm 13 atP stochastic collisions in a closed reactor

```
1: Let M be a multiset of molecules.
2: Let R be a multiset of reactions.
3: while not finished do
4:    Randomly choose a reaction r = (u → sv).
5:    Randomly choose a molecule m .
6:    if m ≡ u then
7:       Remove u from M: M = M \ {u}.
8:       if s = H then
9:          Add v to M: M = M ∪ {v}.
10:      else if s = L then
11:         Send v to all neighbors.
12:      end if
13:   end if
14: end while
```

2. $Z = \{\mathsf{H}, \mathsf{L}$ a set of special symbols: H = HERE, L = LEAVE, and $Z \cap V = \emptyset$;

3. $C \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ a set of interconnections between the m cells;

4. $\sigma_1, \ldots, \sigma_m$ are cells of the form

$$\sigma_i = (w_i, R_i), \tag{6.10}$$

where:

1. $w_i \in V^*$ is the initial multiset of objects within cell i;

2. $R_i, 1 \leq i \leq m$ are finite multisets of *evolution rules* r or *reactions* of the form $r = (u, z, v) = (u \to zv)$, where $u, v \in V^*$ and $z \in Z$:

6.2.6 atP Toy Examples

The atP chemistry is definitely not a really useful chemistry but rather serves as an illustrative purpose only. In the following examples, we will deal with different forms of counters since counters have a long-lasting tradition in our lab.

Example 6.2.1 (atP Single-Cell Counter)
Consider the following single-cell atP *system:*

$$\mathsf{atP} = (V, Z, \sigma_1, C), \tag{6.11}$$

where:

1. $V = \{000, 001, 010, 011, 100, 101, 110, 111\}$;
2. $Z = \{\mathsf{H}\}$; *(here)*
3. $C = \emptyset$ *(no interconnections);*
4. $\sigma_1 = (w_1, R_1)$ *with*
5. $w_1 = \{000\}$;
6. $R_1 = \{000 \to \mathsf{H}001, 001 \to \mathsf{H}010, 010 \to \mathsf{H}011, 011 \to \mathsf{H}100,$
   $100 \to \mathsf{H}101, 101 \to \mathsf{H}110, 110 \to \mathsf{H}111, 111 \to \mathsf{H}000\}$.

It is important to note that the symbols $000, \ldots$ *are inseparable and* not *composed of* 1*s and* 0*s. One could have as well assigned a letters:* $000 = a, 001 = b, \ldots$. *Figure 6.6 illustrates how the single-cell counter chemistry evolves. At the beginning, the single molecule* 000 *is present within the single cell. Eight reaction rules allow to change the molecules in such as way as to allow the chemistry to represent a 3-bit binary counter:* $000 \to 001 \to 010 \to 011 \to 100 \to 101 \to 110 \to 111 \to 000$. *The counter evolves at the speed of the underlying reactor(s) and cannot directly be influenced. By adding multiplicities of each reaction, the reaction speed can however be indirectly increased (increasing the probability that a reaction is chosen).*

■

MATLAB AMB toolbox example:

```
>> cell = initCell({'000'},{'000-H001','001-H010',...
                            '010-H011','011-H100',...
                            '100-H101','101-H110',...
                            '110-H111','111-H000'});
>> cell = runCell(cell);
```

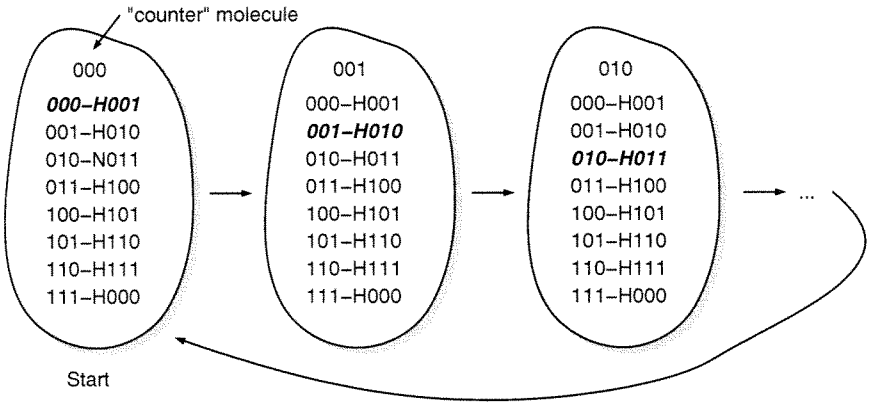

Figure 6.6: Evolution of the atP single-cell counter chemistry according to Example 6.2.1. Reactions that can be applied are in ***bold italics***.

Example 6.2.2 (atP Two-Cell Counter)
Consider the following two-cell atP *system:*

$$\mathsf{atP} = (V, Z, \sigma_1, \sigma_2, C), \tag{6.12}$$

where:

1. $V = \{00, 01, 10, 11, a, c, i\}$;
2. $Z = \{\mathsf{H}, \mathsf{L}\}$; *(here and leave)*
3. $C = \{(1,2),(2,1)\}$;
4. $\sigma_1 = (w_1, R_1)$, $\sigma_2 = (w_2, R_2)$ *with*
5. $w_1 = \{00\}$, $w_2 = \{00\}$;
6. $R_1 = \{00 \to \mathsf{H}01, 01 \to \mathsf{H}10, 10 \to \mathsf{H}11, 11 \to \mathsf{H}c|i, c \to \mathsf{L}c, a|i \to \mathsf{H}00\}$;
7. $R_2 = \{c|00 \to \mathsf{H}a|01, c|01 \to \mathsf{H}a|10, c|10 \to \mathsf{H}a|11, c|11 \to \mathsf{H}a|00a \to \mathsf{L}a\}$.

The two-cell atP *chemistry represents a two-digit counter. A special chemical (c) allows to increment the MSB-counter (most significant bit) that acknowledges is activation by means of the signal (a). The additional chemical i is used to temporary stop the LSB-counter until it receives the acknowledge signal. Figure 6.7 illustrates how the two cells interact. The counter counts as shown in Table 6.2.*

Cell 1 (LSB)	Cell 2 (MSB)	Comment
00	00	
01	00	
10	00	
11	00	$c \rightarrow \mathsf{L}c, a \rightarrow \mathsf{L}a$
00	01	
01	01	
10	01	
11	01	$c \rightarrow \mathsf{L}c, a \rightarrow \mathsf{L}a$
00	10	
01	10	
10	10	
11	10	$c \rightarrow \mathsf{L}c, a \rightarrow \mathsf{L}a$
00	11	
01	11	
10	11	
11	11	$c \rightarrow \mathsf{L}c, a \rightarrow \mathsf{L}a$

Table 6.2: Two-cell counter with its relevant reactions.

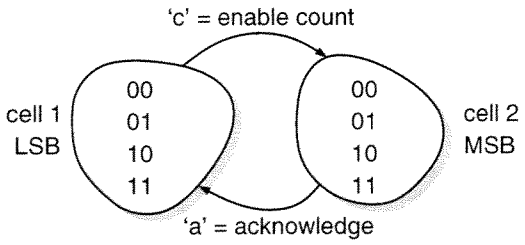

Figure 6.7: A two-digit (one digit per cell) counter that uses an "enable molecule" to active the incrementation of the MSB-digit. An acknowledge signal is sent back. The chemicals in the cells represent the values of each counter.

MATLAB AMB *toolbox example:*

```
>> c1 = initCell({'00'}, {'00-H01', '01-H10', '10-H11',...
                          '11-Hc|i', 'c-Lc', 'a|i-H00'});
>> c2 = initCell({'00'}, {'c|00-Ha|01', 'c|01-Ha|10',...
                          'c|10-Ha|11','c|11-H00', 'a-La'});
>> cells = [c1 c2];
>> conn = [1 2; 2 1];
>> cells = createCellEnsemble(cells,conn);
>> cells = runCellEnsemble(cells);
```

■

Note that one could also have chosen different symbols for representing the counter's values. The symbols used (i.e., 00, 01, ...) have only be chosen to emphasize the fact that they are part of a counter.

6.2.7 Wrap-Up

The main drawback of atP systems is their limitation to direct interconnections between the cells, which are limited to single membrane cells in addition. As the membranes don't play a central role anymore, the system is of a very limited interest for building hierarchical cellular structures.

I the next section I will propose a combination of tissue P systems and classical membrane systems, which seems more appropriate for most applications and which provides many more functionalities.

6.3 aP Membrane Systems

Paun's classical membrane systems possess several properties which complicate hardware implementations, at least distributed implementations. As we have already seen in Section 4.8 (page 178), the following issues are problematic for distributed implementations:

- Paun's membranes are being updated synchronously. This is not only biologically unrealistic (although is was not the goal of P systems to imitate Nature) but problematic for large hardware implementations as clock distribution is usually nontrivial, requires additional resources, and limits scalability.
- The maximum parallel application of the rules as well as the priorities among rules (for a certain class of P systems) are also problematic as both require a "global view" again to decide whether all rules that can be applied have been applied and whether the priorities are respected.

In addition, the underlying amorphous substrate requires some other precautions. The most important issue to consider is probably that amorphous components are inherently unreliable. The membrane system with its chemistry must therfore be tolerant to a certain loss of information, i.e., lost molecules and reactions, faulty membranes, faulty interconnections, etc.

Little is known about asynchronous P systems and this field is actually still active and part of the list of open problems (Problem Q3) [315]. Nevertheless, it is known that asynchronous P systems à la Paun without priorities are not universal [315, p. 78]. However, we will see in this section that it is possible to compute a logical NAND function with aP systems. Since NAND functions form a logical basis, it is basically possible to realize any logical function at the basis of NAND functions. By abstracting away from all practical considerations, one can therefore conclude that an aP membrane system does also allow for universal computation.

Let us now define the basics of an aP membrane system as it shall be used throughout this work. This model is closely inspired by Paun's membrane systems, but was adapted for my rather special purposes. Please keep in mind that the resulting membrane systems is more guided by implementational than by theoretical considerations. The aP membrane system will in Chapter 7 be implemented on a Circuit Amorphous Computer as illustrated in Figure 6.1.

6.3.1 Membranes

Paun's membrane systems usually have a single skin membrane around all the other cells. To the best of my knowledge, *tissue* P *systems* (tP systems) introduced by Martín-Vide *et al.* [250], are represent the only exception: the network of cells is not delimited by a unique skin membrane but is similar to a neural network (with cells seen as neurons). A cell might be considered as an abstraction of a biological cell or neuron, the interconnections as protein channels or synapses.

The membrane structure of an aP systems always requires a skin membrane around a set of cells. This, however, does not prevent from implementing networks of cells as they can easily be delimited by an additional skin membrane in order to respect the aP constraint (Figure 6.8).

An aP membrane system is characterized by the following points:

- The membrane system must have a single skin membrane which represents a border to the environment.
- The *membrane structure* can either be represented by an Euler-Venn diagram, by a parenthesis expression, or a rooted tree (see also Figures 2.27 and 2.28).

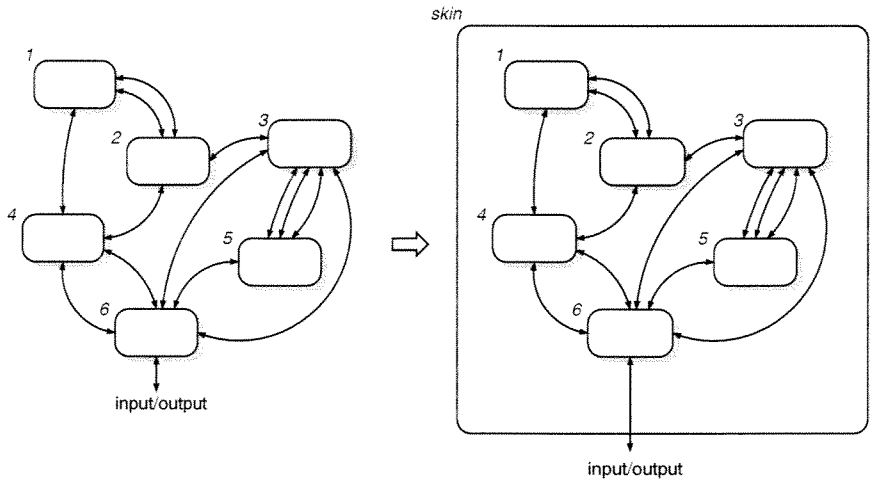

Figure 6.8: A skin membrane can easily be added to a tP system.

- Each cell can only communicate with

  1. cells on the same hierarchical level,
  2. its external enclosing membrane, or
  3. the membranes it contains on the next lower level.

  For example: membrane 4 of Figure 6.9 can send and receive objects from its external membrane 1 (upper-immediate), from 5, 6, and 7 (lower-immediate), and it can also directly send objects to the membranes 2 and 3. However, membrane 4 cannot directly communicate with membranes 8 and 9.

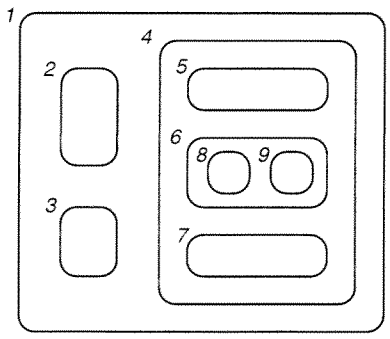

Figure 6.9: Elements of a membrane system represented as a Venn diagram.

Figure 6.10 shows the communication channels which exist between the different membranes. One can imagine that each cellular membrane pro-

duces a certain gradient which is visible by the other cells in the same region, but which does not cross other cellular membranes.

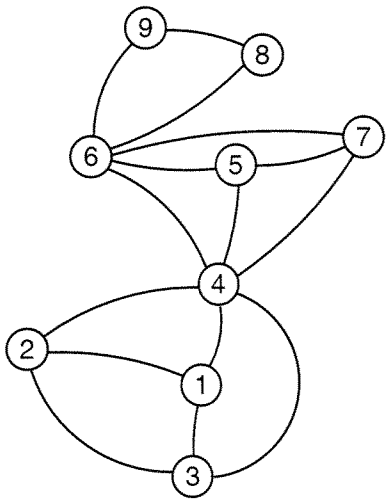

Figure 6.10: Illustration of the communication channels which exist between the different membranes of Figure 6.9. A membrane is represented as a node, a communication channel as an edge.

Note that in a "normal" P system as defined by Paun, membrane 4 *cannot* directly send objects to the membranes 2 and 3! I have introduced this possibility in order to be able to build neural network like membrane structures similar to tP systems. This is not a fundamental modification as it was also possible before to send data via cell 1 from cell 4 to cell 2 or 3.

6.3.2 Molecules and Reactions

An aP membrane can contain a multiset of reactions and molecules. A more formal definition will be provided in Section 6.3.4.

Definition 6.3.1 (aP Molecules)
An aP *molecule is a symbol from an alphabet V or a rule from the set of all possible reactions R^* (see Definition 6.3.2). A set (or string) of molecules forms a multiset of objects over $V \cup R^*$. For example, V might be the alphabet of the 26 letters:*
$V = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$. *The following multisets would be considered valid in that case:*

$$w_1 = a^2bijg^2h^2y$$
$$w_2 = l^2aksu^3$$
$$w_3 = a$$
$$w_4 = u^3$$

In addition, if we assume a set of reactions: $R = \{a \rightarrow \mathsf{H}b, c \rightarrow \mathsf{D}d, u \rightarrow \mathsf{H}a^2\}$, *then the following multisets would also be valid:*

$$w_5 = a^2(a \rightarrow \mathsf{H}b)bijg^2h^2y$$
$$w_6 = (u \rightarrow \mathsf{H}a^2)^2g^4$$
$$w_6 = b(c \rightarrow \mathsf{D}d)(u \rightarrow \mathsf{H}a^2)$$

Note that the rules, which have to be interpreted as molecules, are written within brackets. The possibility of creating rules during a computation has also been discussed in Paun [315, p. 102ff], however only in the context of reactions.

■

An evolution rule of a classical P systems is a rewriting rule of the form

$$u \rightarrow v(v_1, in_i)(v_2, out) \tag{6.13}$$

that replaces the membrane objects u by the objects v, sends the objects v_1 to the lower-immediate membrane with label i, and sends the objects v_2 to the upper-immediate membrane. The aP reactions use a different syntax and offer additional possibilities.

Definition 6.3.2 (aP Reactions)
An aP *reaction is defined as*

$$r = (u, s, v) = (u \rightarrow sv) \tag{6.14}$$

where u, v *are multisets over the symbols of the alphabet* V *and the set* R^* *of all reactions* r, *and where* s *is a* special operator symbol *from* S *as defined in Definition 6.3.3. It is important to note that the multisets* u *and* v *can themselves contain reactions of the above form. Note also that reactions may be present in a membrane in multiple instances (i.e., different* concentrations*), which is not the case for classical* P *systems. The special symbol* H *might be omitted to increase the readability of the reactions, i.e.,* $r = a \rightarrow b^2$. *The same goal is behind the accentuation of*

the main reaction arrow in complicated reactions by means of a longer arrow.

The extension which allows to transform reactions as well, i.e., to consider reactions as molecules as well, allows to do more elegant and complex computations.

Examples of some valid reactions:

$$r_1 = a^2h \longrightarrow ig^2h^2y$$
$$r_2 = a^2h \longrightarrow \mathsf{H}ig^2h^2y$$
$$r_3 = aul \longrightarrow \mathsf{O}b^4(u \rightarrow a^2)^2g^4$$
$$r_4 = b(c \rightarrow d) \longrightarrow \mathsf{D}(u \rightarrow a^2)$$
$$r_5 = b(c \rightarrow d) \longrightarrow \mathsf{D}$$

Note that the multiset v can also be empty. ■

Definition 6.3.3 (aP Special Operator Symbols)
The special operator symbols as illustrated in Table 6.3 are available. $S = \{\mathsf{H}, \mathsf{O}_i, \mathsf{I}_i, SYD, \mathsf{C}_i\}$.

Symbol	Name	Description
H	HERE	Apply the reaction and put molecules in current cell.
O i	OUT$_i$	Move the resulting molecule(s) outside the cellular membrane to cell i, If i is omitted, the molecule(s) will only leave the current membrane.
I i	IN$_i$	Move the resulting molecule(s) to cell i inside the current membrane.
D	DEL	Dissolve the current cell.
C i	CREATE$_i$	Creates new cell with multiset v as contents inside the current region. The membrane number will be set to i.

Table 6.3: aP special operators

■

Note that I also resigned from using rules with several targets (i.e., here, out, in) at the same time in order to simplify things. The same behavior can easily be obtained by using several rules. For example, the rule

$$r = a^2 \longrightarrow \mathsf{H}bc\mathsf{O}_2x\mathsf{I}_4y^2$$

which sends bc to the current membrane, x to the upper-level membrane 2, and y^2 to the lower-level membrane 4 of a given membrane system, can be decomposed into the following rules which yield in the same behavior:

$$\begin{aligned} r_1 &= a^2 \longrightarrow ghi \\ r_2 &= g \longrightarrow \mathsf{H}bc \\ r_3 &= h \longrightarrow \mathsf{O}_2x \\ r_4 &= i \longrightarrow \mathsf{I}_4y^2 \end{aligned}$$

Figure 6.11 illustrates the application of a normal rewriting rule which contains the H special operator symbol. The result of the reaction will simply be written in the current region.

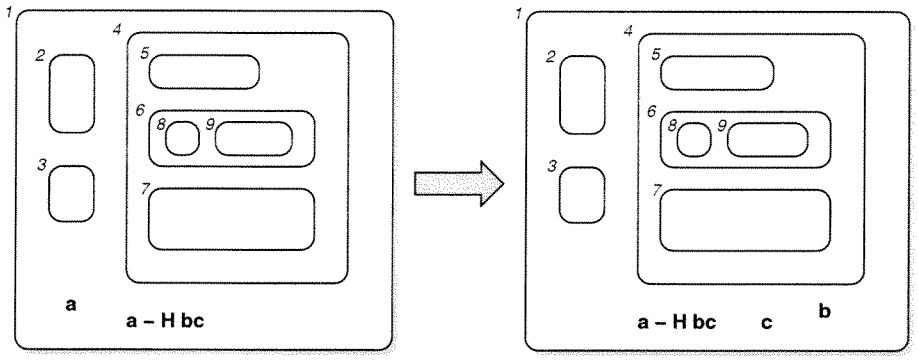

Figure 6.11: Application of a rule that contains the special operator H. The result of the reaction will be written in the current region.

In contrast to the original P system reactions (see Equation 6.13), the aP reaction also allows to send objects to the outer membranes with a specified destination (this was only possible with the inner membranes so far). To be able to send objects to a particular outer (upper-immediate) membrane is necessary in order to respect the communication channels as specified in Section 6.3.1.

Figure 6.12 illustrates the application of a O rule. If such a rule is applied within the skin membrane, the result will be written outside the skin membrane to the environment. Depending on the application and problem, this might be considered as an output of the membrane system. If a O rule without an index i is used, the molecules will simply be moved outside the current membrane. This operator is therefore equivalent to the L operator of atP systems.

If a rule can be applied (i.e., if there are matching molecules available), it will always be applied, independently of whether the destination of an I $_i$ or O $_i$ operator is correct. If there is no cell with the specified index i, the symbols will be lost. The reason for this is again the implementational constraints. Testing whether a cell exists before the rule is applied makes the system way more complicated.

Figure 6.13 illustrates the application of the special operator I $_i$. Both the O and the I operator therefore allow to be applied without an explicit destination, i.e., without a cell index i.

After the D operator has been applied to dissolve the current membrane, all molecules and membranes that were previously present in it, but *not* the reactions, will become elements of the contents of the immediately upper membrane. If the the skin membrane is dissolved, all objects and membranes will be lost. This actually corresponds to a cellular death, which was not allowed in Paun's membrane systems. Figure 6.14 illustrates some examples using the membrane dissolving operator. Algorithm 14 shows the microsteps of the membrane deletion operation. The idea of microsteps was introduced by Paun. A *macrostep* consists in a synchronous update of all membranes in a membrane systems. Each macrostep can be decomposed in to several *microsteps*. A macrostep can only be completed when all microsteps have been completed. It is important to note that the execution of a special operator does not stop the other operators, i.e., if a membrane dissolve operator is applied, other reactions might be executed during the dissolve operator runs. If this is a problem, the user has to make sure that the remaining reactions are inhibited. This might be done by means of a special chemical, for example.

Algorithm 14 aP Membrane Deletion Microsteps

```
1: if reaction is within skin region then
2:     Cellular death: dissolve all membranes and delete the molecules and
       reactions.
3: else
4:     Send molecules to upper-immediate region.
5:     Delete all reactions in the current region.
6:     Dissolve the current membrane.
7:     if current membrane contained further membranes then
8:         Associate these membranes with all their contents to the upper-
           immediate region.
9:     end if
10: end if
```

Finally, if the creation operator C $_i$ is applied, a new cell will be created inside the current region. The number of the membrane will be set to i. The contents of the membrane are given by the multiset v of the reaction $r = u \rightarrow \mathsf{C}v$. Figure 6.15 provides an illustration of the membrane creation.

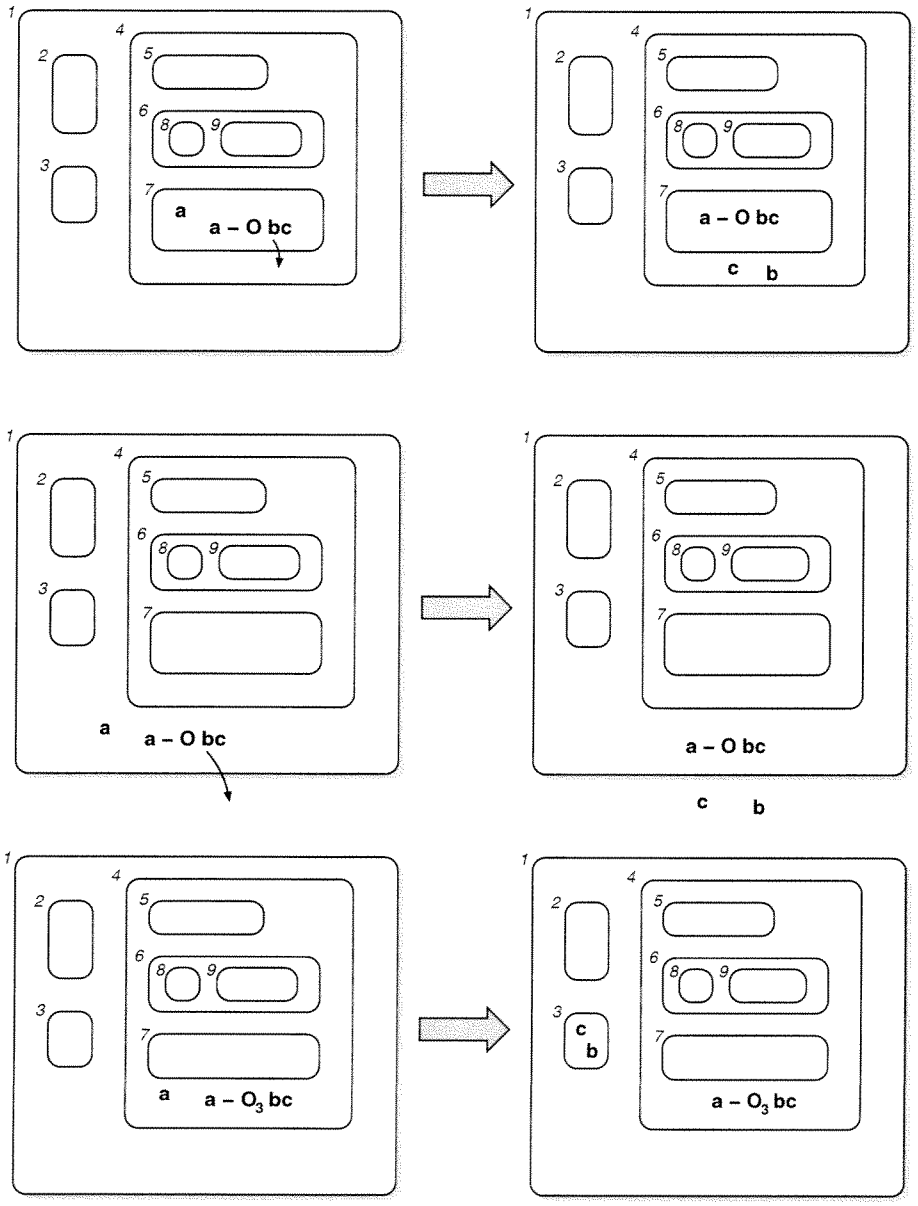

Figure 6.12: Three different application of rules which contain the special operator O_i. The result of the reaction will be written outside the current membrane to region i.

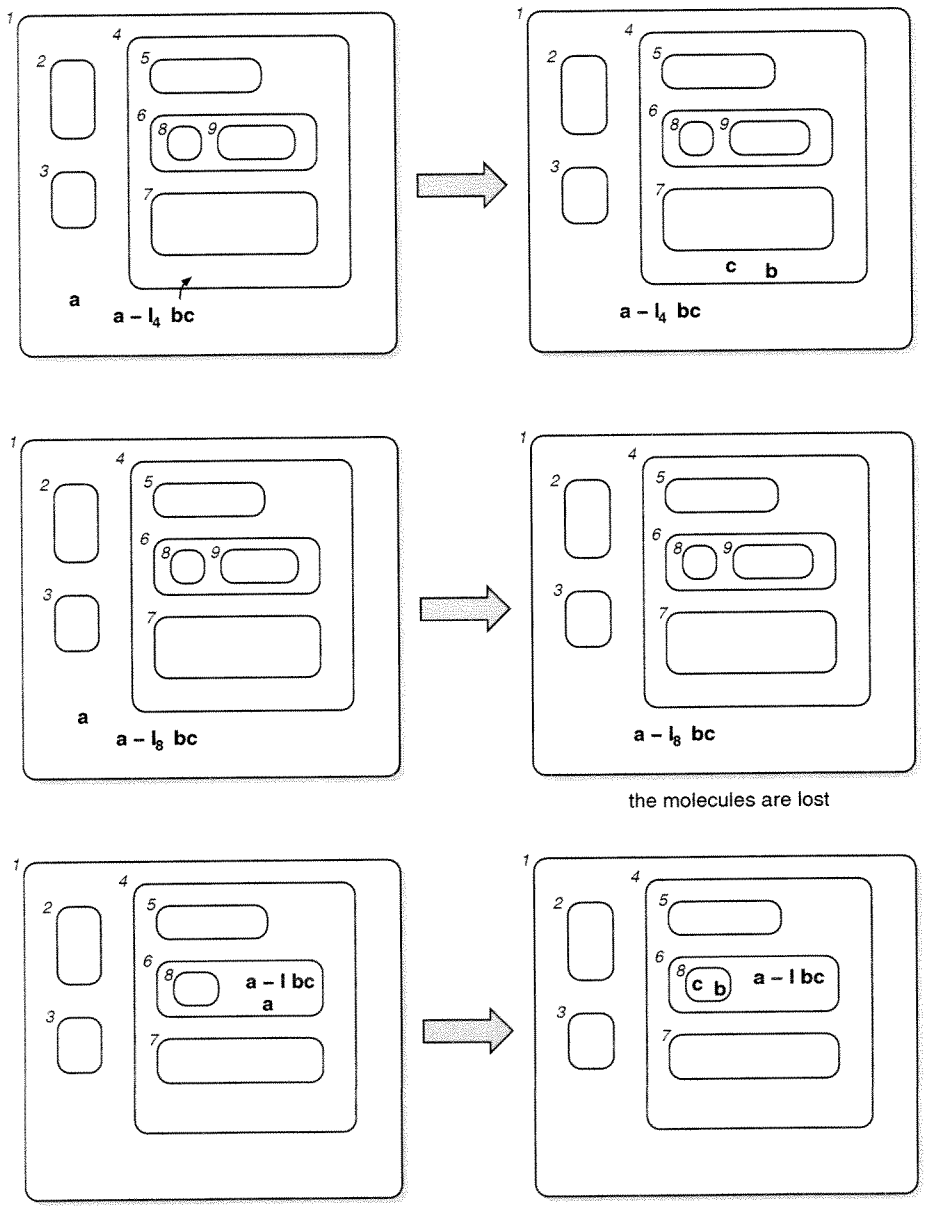

Figure 6.13: Three different application of rules which contain the special operator l_i. The result of the reaction will be written inside the current membrane to region i.

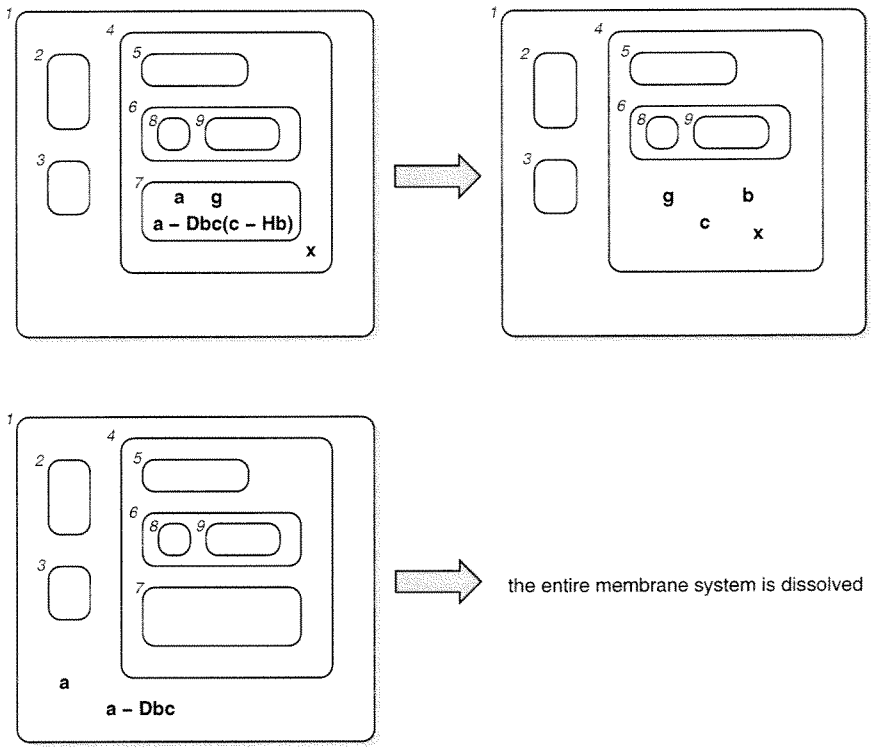

Figure 6.14: Application of a rule which contains the special operator D. The current membrane will be dissolved and the molecules only will become elements of the contents of the immediately upper membrane.

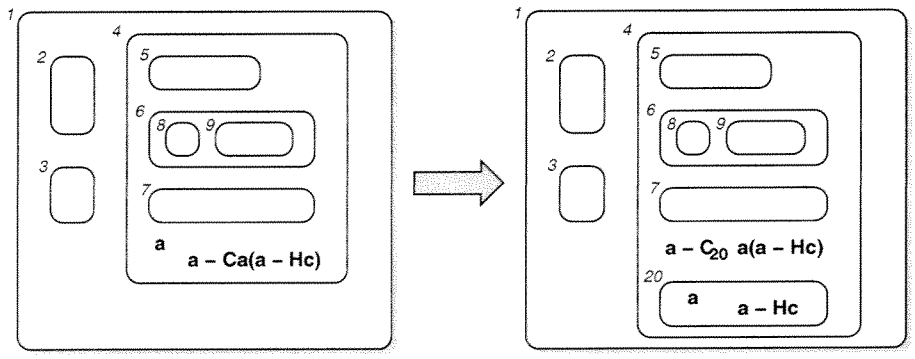

Figure 6.15: Different applications of rules which contain the special operator C_i. A new membrane i will be created outside the current region. If a creation rule is applied within the skin region, the new cell will be created outside the skin membrane.

6.3.3 Reactor Dynamics

The problem with "classical" P systems is that they are difficult to implement on a distributed, minimal, and partially random architecture. However, there are many ways to define the rewriting of the molecules by the reactions. Paun mentions several alternatives [315, p. 208] to the maximal parallel rewriting as usually used. So far, almost everything which has been done in membrane computing was without probabilities. One exception comes by Obtulowicz and Paun [303]. The work presents the introduction of probabilities at various levels in membrane systems. For a certain class of systems — where probabilities are associated with the evolution rules — computational universality is obtained. The main conclusion is that the the models are closer to biology, but also much harder to handle.

I have chosen very simple stochastic reaction dynamics which will be fairly easy to implement by a distributed reactor network, as we shall see later. Compared to the atP reactor dynamics, the algorithm has to be slightly modified as the reactions can itself also be interpreted as molecules. The result is presented in Algorithm 15.

Note that I also avoided any search procedure to search for molecules which match the reactions. Searching for matching molecule-reaction couples would basically only accelerate the process.

The method chosen is not exactly what happens in reality since a chemical system might have more active reactions than others. However, the same effect can be obtained with the above presented algorithm when rules that should be more active are placed inside the reactor in multiple instances.

Note also the each cell might have its own "reaction speed", i.e., a given cell might execute more reactions than another in a given period of time. There is *no* global synchronization signal in aP membrane systems compared to Paun's P systems, where each macrostep was globally synchronized. The developer of the artificial chemistry should bear that in mind and must implement his own synchronization mechanisms if needed.

6.3.4 aP System Formalization

Formally speaking an aP membrane system is a construct

$$\Pi = (V, S, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m), \tag{6.15}$$

where

1. V is an alphabet. Its elements are called *objects* or *molecules*;
2. S is an alphabet of special operator symbols with $S \cap V = \emptyset$;

Algorithm 15 aP stochastic reactor collisions

```
1: Let O be the multiset of molecules and reactions.
2: while not finished do
3:   Randomly choose a reaction r = (u → sv) from O.
4:   Randomly choose an object o (molecule or reaction) from O.
5:   if o ≡ u then
6:     Remove u from O: O = O \ {u}.
7:     Decode and execute special symbol s:
8:     if s = H then
9:       Add v to O: O = O ∪ {v}.
10:    else if s = I_i then
11:      Send v to cell i which lies inside the current membrane.
12:    else if s = O_i then
13:      Send v to cell i which lies outside the current membrane.
14:    else if s = D then
15:      Dissolve current cell.
16:    else if s = C_i then
17:      Create new cell with multiset v as contents inside the current
         region. Set the new membrane region to i.
18:    end if
19:  end if
20: end while
```

3. μ is a membrane structure consisting of m membranes; all membranes (and hence the regions) are invectively labeled with $1, 2, \ldots, m$, where m is called the degree of Π;

4. $w_i, 1 \leq i \leq m$ are strings which represent multisets over V associated with the regions $1, 2, \ldots, m$ of μ;

5. Let $r = (u, s, v)$ be an *evolution rule* or *reaction*, usually written in the form $r = u \rightarrow sv$; let R^* be the set of all possible reactions;

6. s is a special operator symbol from S, u, v are multisets over $V \cup R^*$;

7. $R_i, 1 \leq i \leq m$ are finite sets of *evolution rules* over $V \cup R^*$ associated with each membrane.

An alphabet is a finite non-empty set of abstract symbols. For an alphabet V we denote by V^* the (usually infinite) set of all strings of symbols from V (V^* is the free monoid generated by V under the operation of concatenation, mathematically speaking). The empty string is denoted by λ. The set of special symbol operators S was already detailed in Table 6.3. Note that the result v of an evolution might also be empty.

The definition of the reaction rules is somehow recursive as rules can themselves figure in the multisets u and v, as already shown in Definition 6.3.2. If a rule figures in u or v, it is usually written within parenthesis to avoid ambivalences, e.g., $r = aul \longrightarrow \mathsf{O}b^4(u \rightarrow a^2)^2 g^4$.

If the aP membrane system contains rules of radius greater than one, then we say that Π is a system with *cooperation*, otherwise it's a *non-cooperative system*. A particular class of cooperative systems is that of *catalytic* systems. In a catalytic system, we can either have rules of the form $a \rightarrow sv$, with $a \in V, v \in (V - C)$, or rules of the form $ca \rightarrow scv$, where $c \in C, a \in (V - C)$, v a string over $(V - C)$, and s is a special symbol operator.

As detailed in Section 6.3.3, the rules are applied in a stochastic manner, corresponding roughly to a reactor where molecules and reactions freely float around, but where only one couple might react at each moment in time.

Note that there are no priorities in aP membrane systems.

6.3.5 Object Concentrations and State-Machines

Each object in the multiset (usually denoted by O) of molecules and reactions can be present in multiple appearances. This was already the case for molecules in classical P systems, but not for reactions.

As we will see later, the goal of having multiple appearances of molecules and reactions is basically twofold:

- to facilitate a distributed implementation, and
- to provide redundancy to cope with certain failures.

From the point of view of a real chemistry, multiple appearances of objects do of course also make sense. Whereas the multiplicities of molecules allows for example to "count" things, the multiplicities of reactions basically accelerates the reaction speed (while holding the parameters of the reactor constant). The more identical reactions are present in the reactor, the higher the probability that one of them will be applied.

Let us assume that one would like to gradually create a certain number of chemicals and to hold it constant once the concentration has reached the level of m molecules. The following rules and initial molecules allow to do this (note that I will usually only write down the rules and molecules instead of the entire membrane system formalization):

$$r_1 = a \rightarrow a^2$$
$$r_2 = a^m(a \rightarrow a^2) \rightarrow a^m$$
$$w = a$$

Once the membrane which contains these objects contains m instances of the molecule a, the rule r_1 will be removed and the concentration will remain constant as long as no molecules are removed.

If molecules are being removed, added, or lost, another approach is needed to hold a concentration within a certain tolerance band. It is fairly straightforward to detect an upper limit and to reduce the number of molecules from m to n by means of a reaction of the following form: $r = a^m \rightarrow a^n$. However, it is not possible to detect an lower limit of molecules. Therefore, the only possibility is to slowly and constantly increase the concentration and to decrease it once the upper limit has been reached. The resulting concentration graph will have the form of a sequence of saw-teeth as illustrated in Figure 6.16. The lower limit as set to 8, the upper to 10. One can see that some small peaks exceed the upper limit. The is because the rule r_1 might be applied several times before r_2 (recall that the rule application done at random). As the aP membrane system does not support priorities, the only possibility to avoid the probability to exceed the limit is to is to increase the number of rules r_2 in order to increase to probability that r_2 will be applied before r_1.

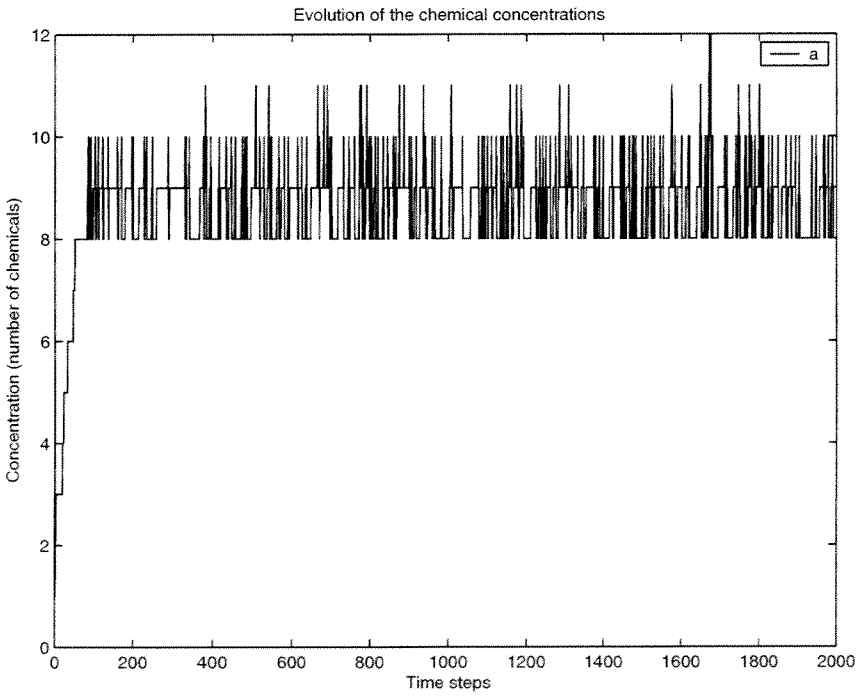

Figure 6.16: Holding a concentration of chemicals constant.

If the system is disturbed, e.g., molecules are added or removed, it will swing back to the desired concentration after a certain amount of time. Fig-

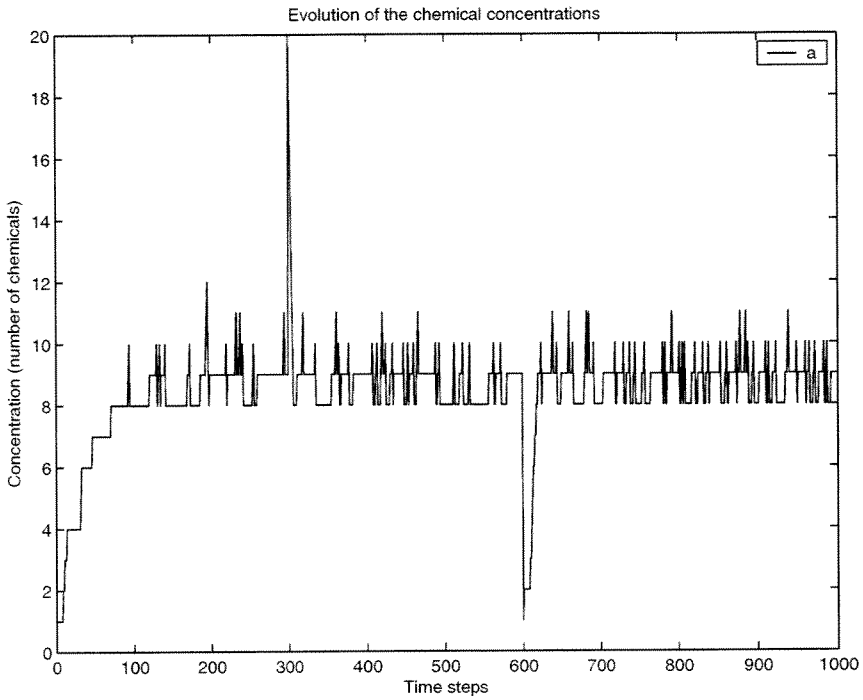

Figure 6.17: The disturbances at time steps 300 and 600 do only briefly alter the chemical concentration.

ure 6.17 shows the concentration of the chemical a which is being disturbed at time steps 300 and 600.

The method for holding an artificial chemical concentration constant can be used as a building block to implement various other systems. One example are chemical state machines: in [191], Hjelmfelt *et al.* developed chemically based clocked finite-state machines, including decoders, binary adders, and stack memory. The state machines are based on a chemical neural network with clocking mechanisms. The state of the system is modeled by chemical concentrations. The same method can be easily used to construct a universal Turing machine [190]. None of these two papers does however deal with the question what logical operations might be performed given a certain chemical reaction mechanism.

In case of a single-cell aP membrane system, a state machine might be implemented fairly straightforward. A state machines is basically composed of three elements: (1) states, (2) transitions, and (3) actions. Figure 6.18 illustrates a sample state machine which shall be implemented by an aP chemistry in the following (Figure 6.19).

In case of a membrane system which is not subjected to errors (i.e., loss or adding of objects), the following rules and objects can be used to realize

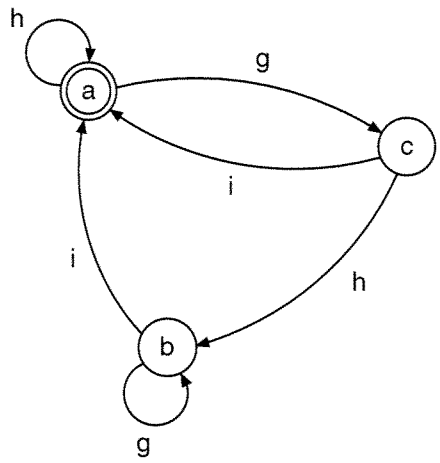

Figure 6.18: A sample state machine with three states (a, b, c) and three inputs (g, h, i) which trigger the state transitions. a is the initial state.

the state machine of Figure 6.18.

$$
\begin{aligned}
r_1 &= ag \rightarrow c \\
r_2 &= ci \rightarrow a \\
r_3 &= ch \rightarrow b \\
r_4 &= bi \rightarrow a \\
w &= a \quad \text{(Initial state)}
\end{aligned}
$$

Each transition is basically represented as a cooperative rule, one condition being the state, the other the input. There is no need to explicitely represent edges which depart from a node and end on the same. Special (output) actions can easily be added to each state by adding rules of the form $state \rightarrow action$. The machine asynchronously changes its state as soon as a new input molecules is available. The state machine can however be synchronized by means of a chemical oscillator as they will be presented in Section 6.3.7. The basic idea is that a chemical, say s, is produced in regular intervals. The rules will then only be applied if this chemical is present. Once applied, the molecule s will be removed until a new one is created in the next interval by the chemical oscillator. The rules would look like this:

$$
\begin{aligned}
r_1 &= ags \rightarrow c \\
r_2 &= cis \rightarrow a \\
r_3 &= chs \rightarrow b \\
r_4 &= bis \rightarrow a \\
w &= a \quad \text{(Initial state)}
\end{aligned}
$$

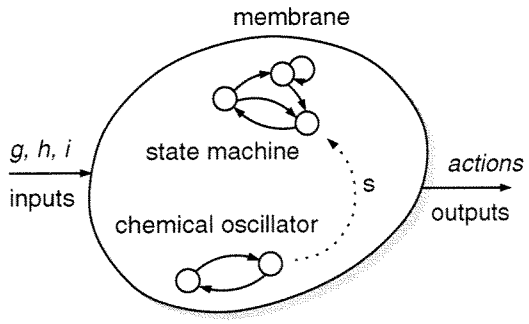

Figure 6.19: An aP membrane system implementing a state machine. The state transitions are triggered by external inputs, e.g. the chemicals g, h, i. If necessary, actions can be associated to states in order to generate output chemicals. A chemical oscillator allows to synchronize the actions of the state machine my means of a chemical s.

The system becomes more complicated when object losses or addings are allowed. The solution I proposed encodes the current state of the state machine in the concentration of a chemical, i.e., if molecule a is present in at least m instances, the machine is considered to be in state a. One has therefore to prevent that several state-representing chemicals are present in a concentration above this level.

$$
\begin{aligned}
r_1 &= a \longrightarrow a^2 \\
r_2 &= a^n \longrightarrow a^m \\
r_3 &= a^m gs(a \to a^2)(a^n \to a^m) \longrightarrow c(a^2 \to a)(c \to c^2)(c^n \to c^m) \\
r_4 &= c^m is(c \to c^2)(c^n \to c^m) \longrightarrow a(c^2 \to c)(a \to a^2)(a^n \to a^m) \\
r_5 &= c^m hs(c \to c^2)(c^n \to c^m) \longrightarrow b(c^2 \to c)(b \to b^2)(b^n \to b^m) \\
r_6 &= b^m is(a \longrightarrow a^2)(a^n \longrightarrow a^m) \longrightarrow a(b^2 \to b)(a \to a^2)(a^n \to a^m) \\
w &= a^m bc \qquad \text{(Initial state} = a)
\end{aligned}
$$

The basic idea behind this implementation is the following: the concentration of a state is hold constant by means of the "saw-tooth" method seen above. For the initial state a, the rules r_1 and r_2 will do this. In case of the an input g, the state will have to change to c. This is done be adding two rules that hold the concentration of c constant, by adding a rule that removes the molecules a, and by removing the rules which held a constant. Obviously, the period of the synchronizing chemical s must be much slower than the time required to increase the chemical concentration to the lower-limit m, otherwise the machine will not function properly. Initially, m instances of the molecule a are available within the membrane as well as one instance

of b and c which are required for increasing their number in case we change the state of the machine.

Note that the machine is tolerant to an important number of failures on the level of the molecules: the concentration of the chemicals might be disturbed as we have seen in Figure 6.17 and the machine continues to function properly. It is possible to remove all molecules but one and the concentration will be re-established quickly. However, the machine will only operate correctly if this happens between two "clock impulses" s, otherwise the transition cannot be take because not enough, i.e. $< m$, molecules are present within the cell.

6.3.6 Simulating Boolean Circuits

Boolean logical functions are at the basis of any modern computer. It would of course be interesting to be able to implement logical functions by means of membrane systems.

Ceterchi and Sburlan [64] recently presented a study which proposed several different implementations. They first implemented the necessary gates and then showed that they can be combined to a certain class of more complex circuits. The solution which uses cooperative rules is trivial, but the P system with catalysts or weak priorities become very complex. There exist actually much simpler membrane systems with mobile catalysts. The membrane systems as presented in Example 6.3.1 — a AND and a NOT gate for classical P systems— illustrate this[1]. Figure 6.20 shows the evolution of the membrane system for possible all four input cases.

Example 6.3.1 (AND and NOT with mobile catalysts)

$$\Pi_{AND} = (V, C, \mu, w_1, w_2, R_1, R_2), \tag{6.16}$$

where

1. $V = \{0, 1, d, e, n, x, z, and\}$,
2. $C \subseteq V = \{e, d, and\}$ *is the set of mobile catalysts,*
3. $\mu = [_1[_2]_2]_1$,
4. $w_1 = \{d, e\}$, $w_2 = \{and, input_1, input_2\}$,
5. $R_1 = \{xand \to and_{in}, e0 \to e_{in}z, ez \to 0_{out}, d1 \to d_{in}, dn \to d1_{out}\}$, $R_2 = \{0and \to 0_{out}and_{out}, 1and \to 1_{out}and_{out}, e0 \to e_{out}x_{out}, e1 \to e_{out}x_{out}, d0 \to d_{out}z_{out}x_{out}, d1 \to d_{out}n_{out}x_{out}\}$.

[1] I am indebted to Petreska Biljana for drawing my attention to this.

$$\Pi_{NOT} = (V, C, \mu, w_1, w_2, R_1, R_2), \tag{6.17}$$

where

1. $V = \{0, 1, d, e, n, x, z, and\}$,
2. $C \subseteq V = \{n, x\}$ *is the set of mobile catalysts,*
3. $\mu = [_1[_2]_2]_1$,
4. $w_1 = \{x\}$, $w_2 = \{n, input\}$,
5. $R_1 = \{nx \rightarrow n_{in}x\}$, $R_2 = \{0n \rightarrow 1_{out}n_{out}, 1n \rightarrow 0_{out}n_{out}\}$.

The inputs must be sent in both membrane systems to region 2. In combining the two membrane systems, one can basically simulate any logical function as AND and NOT functions together form a logical basis.

■

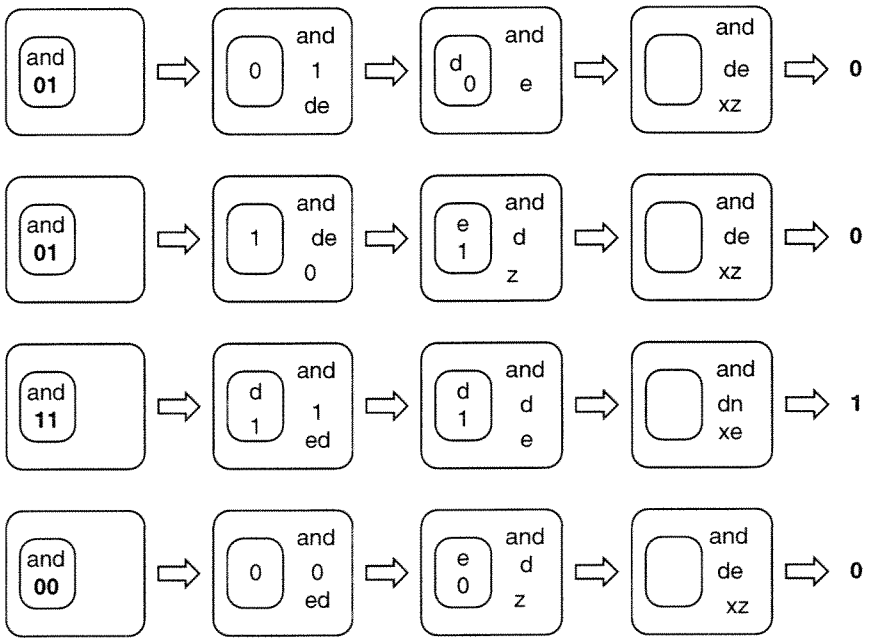

Figure 6.20: Evolution of the classical P system with mobile catalysts. The membrane system computer the logical AND function, is self-synchronized, and fairly simple.

Despite the fact that there are much simpler systems than proposed by Ceterchi and Sburlan, there exist some further and major drawbacks with their approach:

- The method does only allow to compute boolean functions which can be put in the form of a rooted tree (the root being the output, the leaves the inputs, the nodes the boolean functions).
- It is an unsolved problem how the input values will reach the appropriate input membranes, which is usually always located at the lowest level of the membrane system.

Using an aP membrane system, the most straightforward way to implement boolean functions is to use cooperative rules of the following form:

- $R_{NOT} = \{0 \rightarrow \mathsf{O}1, 1 \rightarrow \mathsf{O}0\}$
- $R_{AND} = \{00 \rightarrow \mathsf{O}0, 01 \rightarrow \mathsf{O}0, 11 \rightarrow \mathsf{O}1\}$
- $R_{NAND} = \{00 \rightarrow \mathsf{O}1, 01 \rightarrow \mathsf{O}1, 11 \rightarrow \mathsf{O}0\}$
- $R_{OR} = \{00 \rightarrow \mathsf{O}0, 01 \rightarrow \mathsf{O}1, 11 \rightarrow \mathsf{O}1\}$

The rules might also be extended to more input variables. Note that it is not necessary that both values arrive at the same time in the cell as the cooperative rule can only be applied when the two input molecules are available. The result is then sent outside the cellular membrane.

In order to computer the boolean function as shown in Figure 6.21, a membrane system as illustrated in Figure 6.22 might be used. The membrane structure is very similar to the structures proposed in [64], but without any artificial complication with catalysts, etc. As already said above, the main challenge consists in sending the input values to the correct membranes at the initialization. This might for example be realized by means of special rules which only transport the objects from the skin membrane to the inner membranes. In order to avoid that a computation starts when the objects are moved to the inner membranes, the need to be named differently (e.g., a and b instead of 0 and 1) and be transformed to their original at the destination. Note that the aP membrane system solution is completely self-timed and synchronized, i.e. a computation starts as soon as the input objects are present.

In addition, I would like to mention some other important points:

1. For aP membrane systems, the underlying structure of the logical circuit does not necessary to be in the form of a rooted tree. The reason is that it is possible to directly (i.e., addressed by O_i) send objects to an upper-immediate membrane instead of only sending them outside the current membrane. This basically allows to have branchings in the tree and to have several outputs.

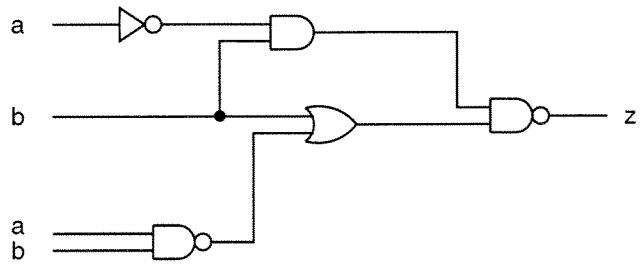

Figure 6.21: A boolean logical function with two inputs, a and b, and one output z. The circuit can be represented as a rooted tree.

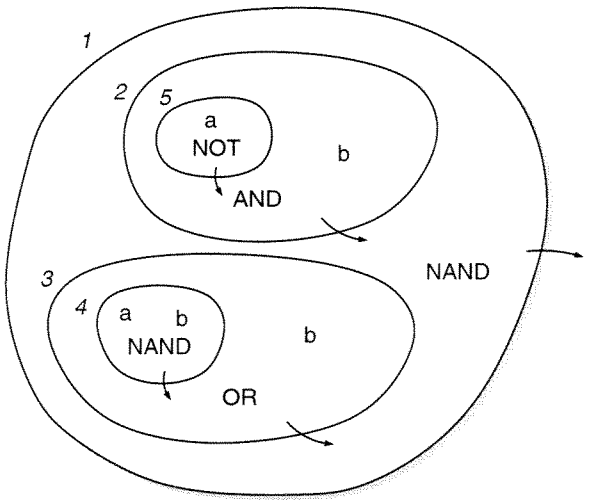

Figure 6.22: An aP membrane system which implements the boolean function of Figure 6.21. a and b represent the binary input values of the circuit.

2. In order to provide fault-tolerance based on redundancy, the molecules might be available in multiple instances. The rules would be of the form $0^m1^m \longrightarrow 1^m$. This is of course a very interesting feature, especially for uncertain environments.

3. As we have seen in Section 2.2.1, the NAND function forms a logical basis and basically allows to implement any logical function. As we have seen above, a NAND function can very easily be implemented with our membrane system and it can therefore be concluded that it is theoretically possible to realize any logical function from a NAND membrane cell by assembling them cleverly (which might of course be very challenging in practice).

6.3.7 Clocking and Oscillators

Although there is no global clock and synchronization in the aP membrane systems (in contrast to the large majority of P systems), it is quite often necessary to dispose of a regularly oscillating signal which might then be used to synchronize events or to serve as a local time emitter.

In then neural-network-based state machine implementation as presented by Hjelmfelt *et al.* in [191], each chemical neuron is updated in discrete time steps. They implemented the necessary discreteness of time and the synchronization of the state changes by means of chemical, autonomous oscillating catalyst ε. The concentration of ε is assumed to be very small, except during an short period of time (compared with the oscillator period). This is a common behavior of nonsiusoidal chemical oscillators [131]. The catalyst ε is then interacts with other chemicals of, for example with two chemical neurons A_j and $A'j$, where and a rapid equilibration only occurs during a short period of time when then concentration of ε is high.

$$A_j \overset{\varepsilon}{\rightleftharpoons} A'j \tag{6.18}$$

Implementing chemical oscillators in P systems is pretty straightforward. I successfully tested a single-cell system which is based on two oscillating chemicals coupled together by two catalysts. The following rules are required:

$$\begin{aligned}
r_1 &= xb^2 \longrightarrow x \\
r_2 &= xb \longrightarrow x \\
r_3 &= ya^2 \longrightarrow y \\
r_4 &= ya \longrightarrow y \\
r_5 &= xa \longrightarrow xa^2 \\
r_6 &= yb \longrightarrow yb^2 \\
r_7 &= xa^{10} \longrightarrow ya^{10} \\
r_8 &= yb^{10} \longrightarrow xb^{10}
\end{aligned}$$

The reactions r_1 to r_4 are used to reduce the number of molecules, whereas r_5 and r_6 are responsible for increasing the concentration. In order to decrease the concentration faster and till zero, there are two reactions. Reactions r_7 and r_8 detect when at least ten molecules are available and inverse the process.

The period of the oscillation can be changed by changing rules r_7 and r_8 or by changing the increase and decrease rules.

In order to function correctly, the system must have one molecule of a and the maximum molecules of b (or the inverse) together with the catalyst x. For example:

$$w_1 = a$$
$$w_2 = b^{10}$$
$$w_3 = x$$

In MATLAB, this looks as following:

MATLAB AMB toolbox example:

```
c = initCell(...
  {'a','b','b','b','b','b','b','b','b','b','b','x'},...
  {'x|b|b-Hx','x|b-Hx','x|a-Hx|a|a',...
  'x|a|a|a|a|a|a|a|a|a|a|a-Hy|a|a|a|a|a|a|a|a|a|a|b',...
  'y|b|b|b|b|b|b|b|b|b|b|b-Hx|b|b|b|b|b|b|b|b|b|b|a',...
  'y|b-Hy|b|b','y|a|a-Hy','y|a-Hy'});
steps = 1000;
plotChemicals(c,{'a','b'},steps);
```

Figure 6.23 and 6.24 show two views of the oscillating chemicals. Such an oscillating chemical might easily be used to synchronize other events by using a rule that "measures" a certain concentration and then triggers an event. The following rule would for example do this: $a^6 \rightarrow a^6 e$. When at least six a's are present in the chemistry, the molecule (event) e is created.

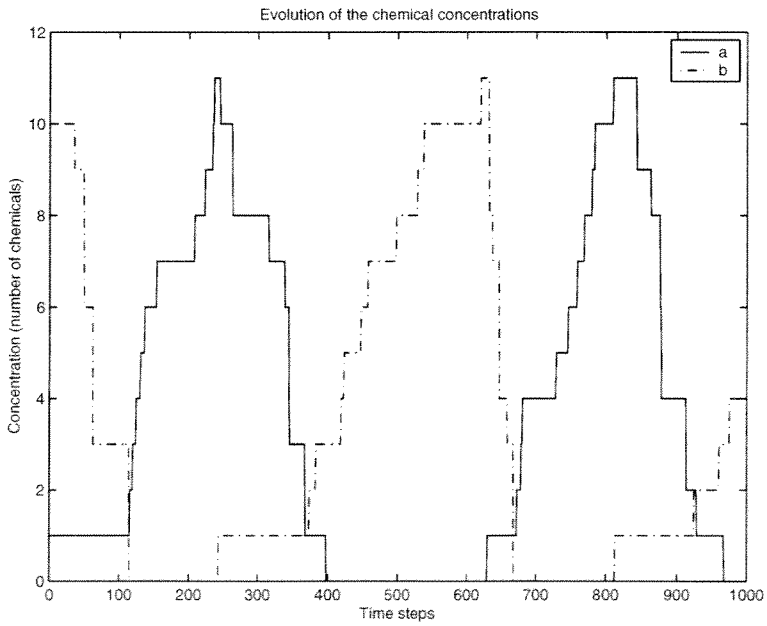

Figure 6.23: A chemical oscillator based on two chemicals a and b and on two catalysts x and y. Time steps: 1000.

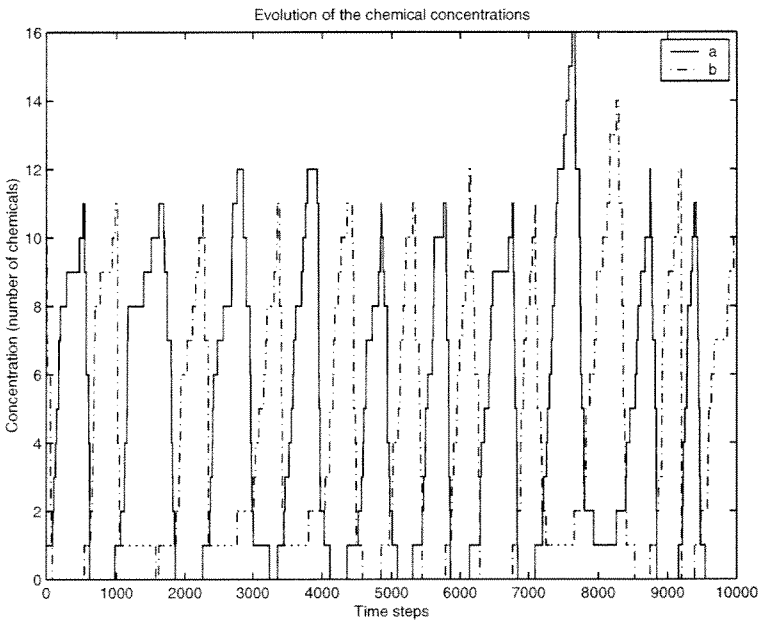

Figure 6.24: A chemical oscillator based on two chemicals a and b and on two catalysts x and y. Time steps: 10000.

6.4 aB Membrane Systems

6.4.1 Introduction

The aP membrane system presented in Section 6.3 represents a special class of membrane systems, which was proposed in view of an implementation on an amorphous computer.

As one of the goals of this thesis was to unify membrane systems, amorphous computers, and a computational version of blending, we will now proceed to the definition of a further membrane system: the aB membrane system.

aB membrane systems are basically an extension of aP membrane systems. The newly added features will allow to combine membrane systems together with the computational C-Blending method as presented in Section 5.6.

Figure 6.25 illustrates how blending in an aB membrane systems is intended to work. The concepts envisions to maintain a population of cells embedded within a special cell of the membrane system. C-Blending will then only take place within this special population. Imagine for example a robotic controller which is composed of two parts embedded within one and the same membrane system: one part is fixed and deals with inputs and outputs, does some pre-processing, etc. The other part consists in a population of membranes which are constantly blended in order to obtain new calls which are better adapted to some give task, as for example to avoid obstacles. Figure 6.26 shows how such a robotic control system might look like. The population of cells within membrane 4 constantly "blends" new cells in order to improve the robots behavior.

I will limit the work in this section to a blend between two cells of a given population. The blend will be initiated by a special rule and the two participating cells will then be melt together to form a new cell. The process of blending and controlling entire populations of cells will have to be addressed in future work.

Note that neither C-Blending nor aB membrane systems do currently allow to make blends between cells of different membrane systems, i.e., systems separated by a skin membrane, and between cells that do not lie in the same membrane region and on the same hierarchical level.

As we have seen in Section 5.6, C-Blending was introduced by using blending between the reactions (i.e., the "program") only. The molecules (i.e., the "data") did not take part in the process. Although this shall not be done here, one could however easily imagine the same mechanisms applied to molecules too.

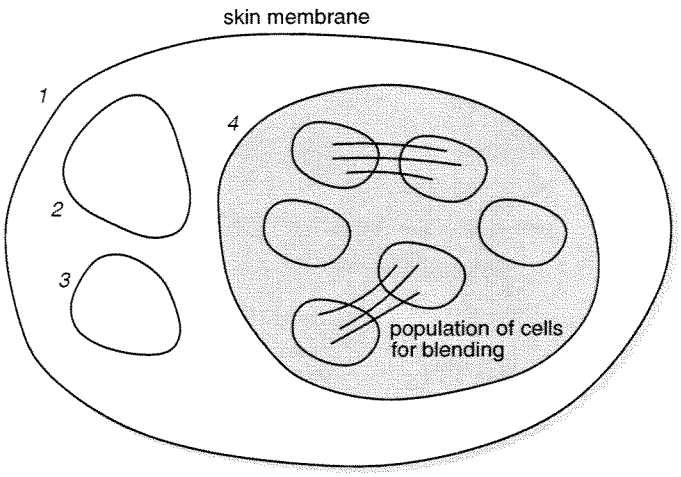

Figure 6.25: The idea of an aB membrane system is to maintain a small population cells for blending within a special cell (i.e., cell 4).

6.4.2 Membranes

The same membranes structures as for aP membrane systems are allowed and supported in aB systems.

An aB membrane system is characterized by the following points:

- The membrane system must have a single skin membrane which represents a border to the environment.
- The *membrane structure* can either be represented by an Euler-Venn diagram, by a parenthesis expression, or a rooted tree (see also Figures 2.27 and 2.28).
- Each cell can only communicate with
  1. with cells on the same hierarchical level,
  2. with its external enclosing membrane, or
  3. the membranes it contains on the next lower level.

An example is provided in Figure 6.25. Note that the cells within membrane four can send objects to the region which contains cell two and three, but also directly into the region of cell two and three.

6.4.3 Molecules

The aB molecules are identically defined as the molecules of aP membrane systems. Definition 6.4.1 provides some examples.

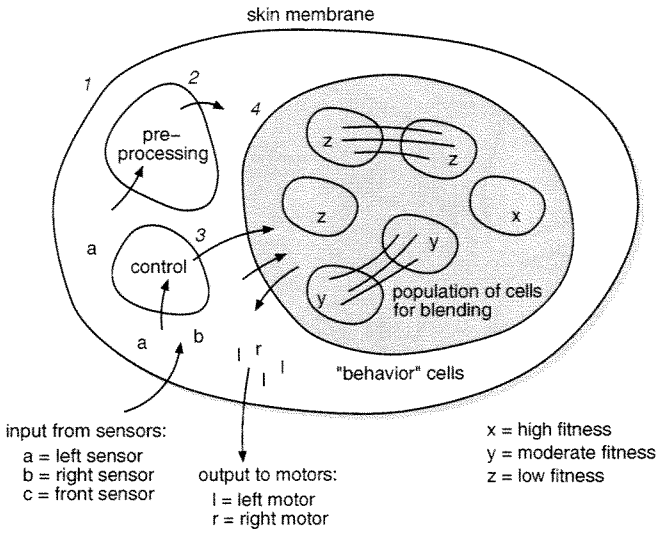

Figure 6.26: A robotic control membrane system might look like this. The population of cells within membrane 4 constantly "blends" new cells in order to improve the robots behavior. The fitness of each cell might be encoded in a symbol and updated by a supervisor/control cell.

Definition 6.4.1 (aB Molecules)
An aB *molecule is a symbol from an alphabet V or a rule from the set of all possible reactions R^* (see next section). A set (or string) of molecules forms a multiset of objects over $V \cup R^*$. For example, V might be the alphabet of the 26 letters:*
$V = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$. *The following multisets would be considered valid in that case:*

$$w_1 = a^2bijg^2h^2y$$
$$w_2 = l^2aksu^3$$
$$w_3 = a$$
$$w_4 = u^3$$

In addition, if we assume a set of reactions: $R = \{a \rightarrow \mathsf{H}b, c \rightarrow \mathsf{D}d, u \rightarrow \mathsf{H}a^2\}$, *then the following multisets would also be valid:*

$$w_5 = a^2(a \rightarrow \mathsf{H}b)bijg^2h^2y$$
$$w_6 = (u \rightarrow \mathsf{H}a^2)^2g^4$$
$$w_6 = b(c \rightarrow \mathsf{D}d)(u \rightarrow \mathsf{H}a^2)$$

Note that the rules, which have to be interpreted as molecules, are written within brackets. The possibility of creating rules during a compu-

tation has also been discussed in Paun [315, p. 102ff], however only in the context if reactions.

■

6.4.4 Reactions

There are some important differences between the definitions of aP and aB reactions. In order to be able to implement the computational C-Blending approach, a class of special reactions is introduced. The reason for this was the insight that one would always require "normal" reactions besides the reactions which will take part in a blending process. The non-blending reactions would for example be used to normally operate a membrane system.

In summary, an aB membrane systems can contain two types of reactions:

1. Reactions as already defined for aP membrane systems in Section 6.3.2. These reactions will *not* take part in any blending operation.
2. Special reactions for blending, which might however also be used for any other "task".

Since normal reactions were already defined in Section 6.3.2, I will only focus on the blending reaction in this section.

Definition 6.4.2 (aB Blending Reactions)
An aP *blending reaction is defined as*

$$r = (u, a, s, v) = (u \rightarrow_a sv) \tag{6.19}$$

where u, v are multisets over the symbols of the alphabet V and the set R^* of all reaction r, where s is a special operator symbol *from S (see Definition 6.4.3), and where a represents the activity of the reaction.*

Again, note that the multisets u and v can themselves contain reactions of the above form. Note also that reactions may be present in a membrane in multiple instances (i.e., different concentrations*), which is not the case for classical* P *systems.*

The activity a of an aB *blending reactions is computed in the following way:*

1. *All activities is set to zero at the creation of a given membrane system.*
2. *Each time a new blending reaction is created, its activity is also set to zero.*

3. *Each time a blending reaction is successfully applied, the activity counter a is incremented by one.*

Some valid reactions:

$$r_1 = a^2h \longrightarrow_{12} ig^2h^2y$$
$$r_2 = a^2h \longrightarrow_0 \mathsf{H}ig^2h^2y$$
$$r_3 = aul \longrightarrow_1 \mathsf{O}b^4(u \rightarrow a^2)^2g^4$$
$$r_4 = b(c \rightarrow d) \longrightarrow_5 \mathsf{D}(u \rightarrow a^2)$$

■

Let us assume a cell which contains the following multisets of reactions and molecules at initialization:

$$r_1 = a \longrightarrow_0 \mathsf{H}x$$
$$r_2 = b \longrightarrow_0 \mathsf{O}y^2$$
$$w = \{a, h, j\}$$

Each time one of the two reactions is applied by the reaction algorithm, the activity counter will be updated, e.g., when r_1 is used to rewrite an a-molecule, it will become $r_1 = a \longrightarrow_1 \mathsf{H}x$ after the first application.

The attentive reader did hopefully already anticipate that this activity measure will be used to establish the activity and similarity based pairings between the rules as detailed in Section 5.6.4.

The blending process requires one more special operator symbols in addition to the already existing aP operators. All operators are are listed below in Table 6.4.

Definition 6.4.3 (aB Special Operator Symbols)
The special operator symbols as illustrated in Table 6.4 are available: $S = \{\mathsf{H}, \mathsf{O}_i, \mathsf{I}_i, SYD, \mathsf{C}_i, \mathsf{B}_i\}$.

■

I will only concentrate on the newly added operator here. For all other special symbols, see Section 6.3.2.

Let us first take a look at the microsteps of the blending operator B. Algorithm 16 illustrates the different steps. Assume the membrane systems as depicted in Figure 6.27. Membrane 5 contains a reaction with the special operator symbol B_6, i.e., when this reaction is applied, blending will be

Symbol	Name	Description
H	HERE	Apply the reaction and put molecules in current cell.
O $_i$	OUT$_i$	Move the resulting molecule(s) outside the cellular membrane to cell i, If i is omitted, the molecule(s) will only leave the current membrane.
I $_i$	IN$_i$	Move the resulting molecule(s) to cell i inside the current membrane.
D	DEL	Dissolve the current cell.
C $_i$	CREATE$_i$	Creates new cell with multiset v as contents inside the current region. The membrane number will be set to i.
B $_i$	BLEND$_i$	Blend the current cell with cell i.

Table 6.4: aB special operators

initialized together with membrane 6. Note that the reaction $x \rightarrow \mathsf{B}_6$ does not necessary be present within the cell from the beginning. One might for example imagine that a rule from an external (controller) cell of the following form send the blending rule inside the cell which shall start the blending process: $a \rightarrow I_5(x \rightarrow \mathsf{B}_6)$. This rule might be present beforehand in region 4.

Once the blending reaction is applied, the first step consists in merging together the two cells and to form one single cell. Thereby, the cell number of the cell which initiated the blending process is kept. All reactions are put together in then new compartment. Figure 6.28 illustrates this microstep.

The next microstep consists in removing all non-blending reactions, i.e, reactions which did not record their activity, and all the molecules. The membrane now only contains blending reactions with an activity measure (see Figure 6.28). Now there comes a difference to the C-Blending method of Section 5.6: as the rules do all "float" in the same membrane region now, they can no longer be distinguished from which cell the came.

However, this is rather good than bad news as it allows to potentially create mappings between two reactions from all and the same cell as well, which opens the space of possible solutions.

The next microstep consists in finding a "good" mapping between pairs of reactions. The mapping is, as explained in Section 5.6, once more based on the rule similarity and the rule activity. Whether this really makes sens has to be decided for each problem separately.

Finally, the established pairings are at the basis of the newly blended reactions according to the method of C-Blending, i.e., a random selection of molecules from each side and rules is considered to build the new rule. Again and of course, many other variant would be possible. In the last microstep,

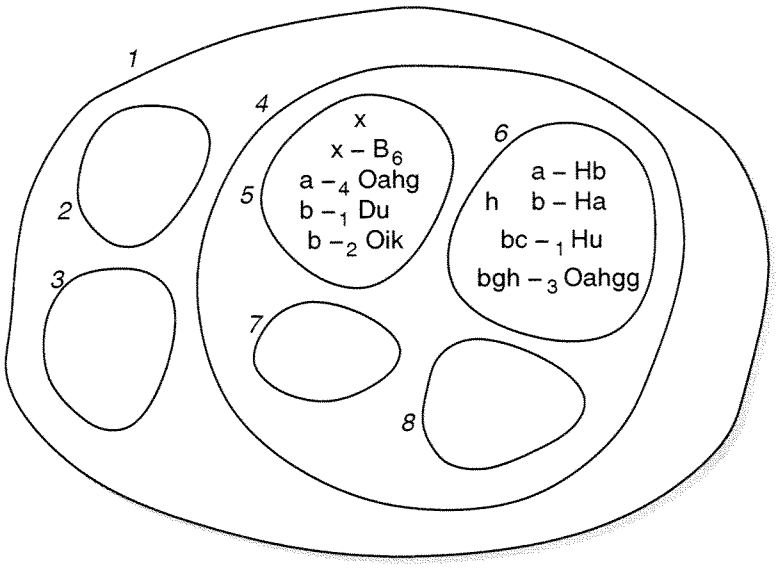

Figure 6.27: The membrane system contains a set of cells located within membrane 4 which will be used for blending. Membranes 5 and 6 contain several blending reactions in addition to the normal reactions.

the activities of each rule are set of 0 again. This completes the macrostep and the cell becomes serviceable again.

Algorithm 16 summarizes the different microsteps, which together form the B-macrostep.

Algorithm 16 aB Membrane Blending Microsteps

1: Merge together the cell which initiated blending and the cell specified in the B-rule.
2: Remove all non-blending reactions and molecules.
3: Create one-by-one initial reaction mapping according to the rule similarity and activity (see Section 5.6.5).
4: Improve the mapping.
5: Blend new rules.
6: Set the activity to 0 for all rules.

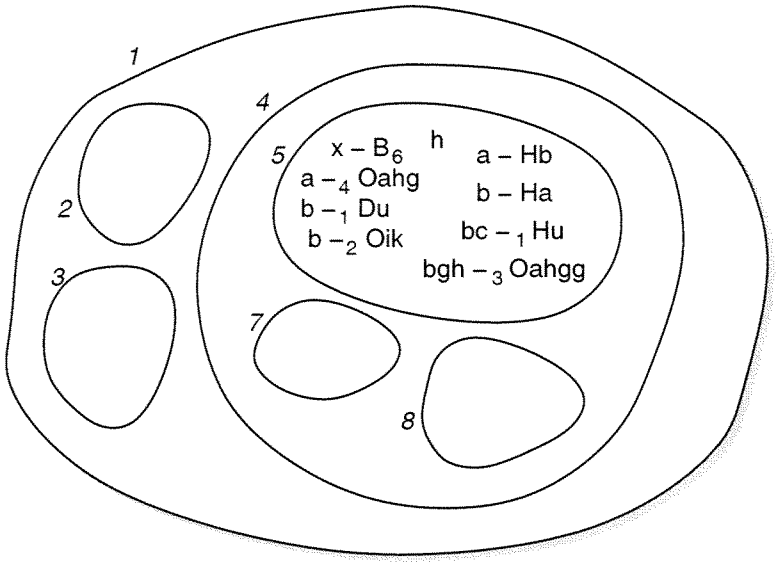

Figure 6.28: The first microstep consists in merging the two cells together.

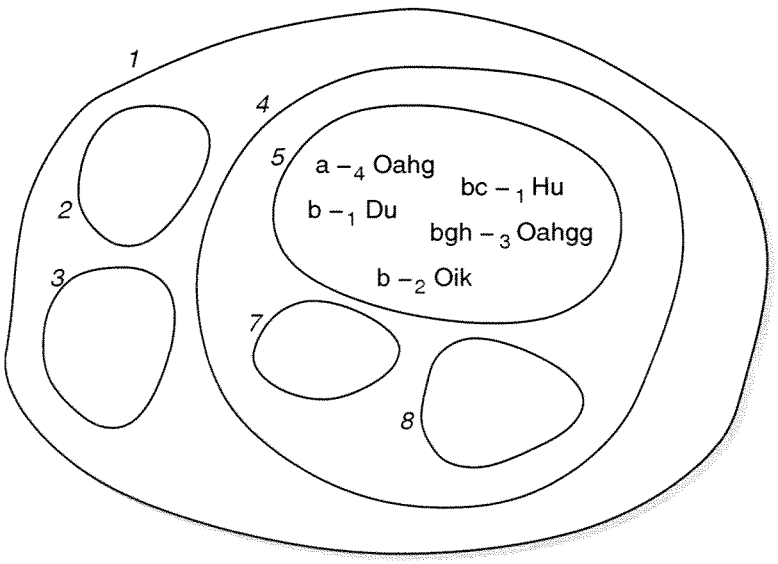

Figure 6.29: All non-blending reactions and molecules are removed from the membrane.

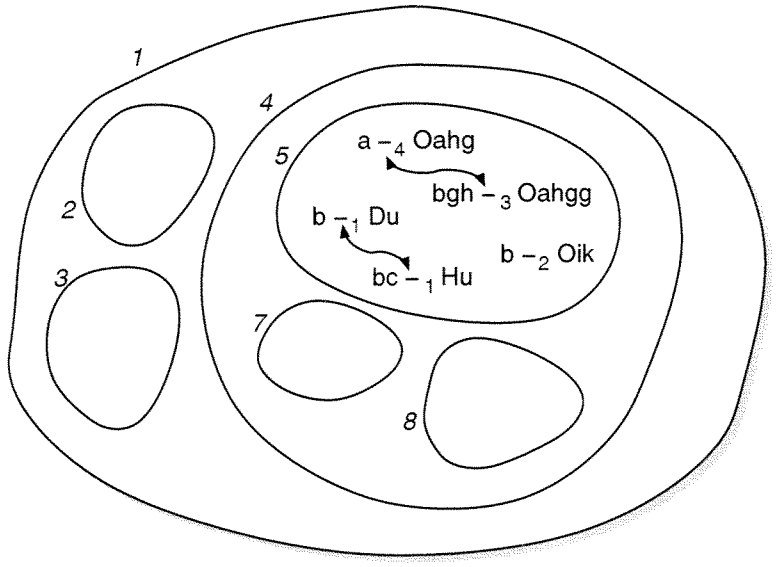

Figure 6.30: A one-by-one mapping between the reactions will be established. The criteria for choosing reactions is based on the similarity and the activity of the participating rules.

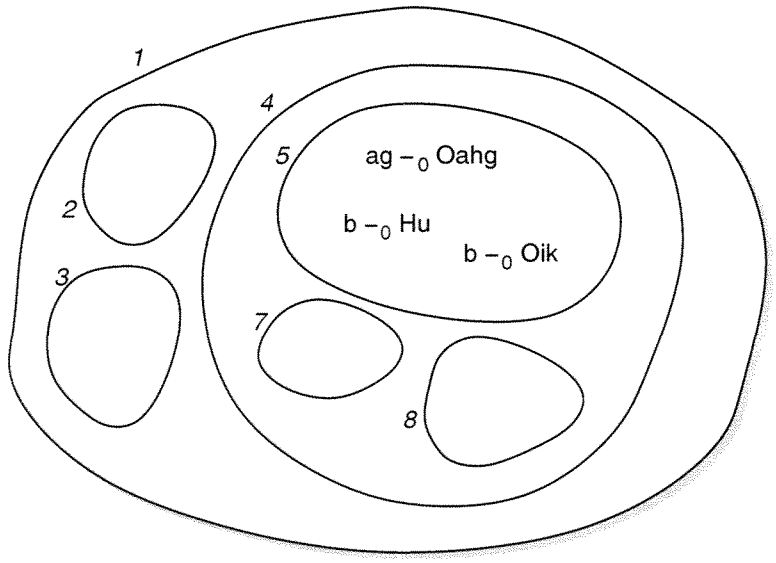

Figure 6.31: Based in the mapping previously established between the reactions, the new rules are blended and their activity is set to 0.

6.4.5 Reactor Dynamics

The same simple stochastic reaction dynamics as for aP systems are valid for aB membrane systems. The basic algorithm is presented in Algorithm 17. Note that the only modifications with regards to the aP algorithm consist in the inclusion of the new special operators required for blending.

Algorithm 17 aB stochastic reactor collisions

Let O be the multiset of molecules and reactions.
while not finished **do**
  Randomly choose a reaction $r = (u \to sv)$ or $r = (u \to_a sv)$from O.
  Randomly choose an object o (molecule or reaction) from O.
  if $o \equiv u$ **then**
    Remove u from O: $O = O \setminus \{u\}$.
    Decode and execute special symbol s:
    if reaction has an associated activity **then**
      Increment the activity: $r = (u \to_{a+1} sv)$.
    end if
    if $s = \mathsf{H}$ **then**
      Add v to O: $O = O \cup \{v\}$.
    else if $s = \mathsf{I}_i$ **then**
      Send v to cell i which lies inside the current membrane.
    else if $s = \mathsf{O}_i$ **then**
      Send v to cell i which lies outside the current membrane.
    else if $s = \mathsf{D}$ **then**
      Dissolve current cell.
    else if $s = \mathsf{C}_i$ **then**
      Create new cell with multiset v as contents inside the current region. Set the new membrane region to i.
    else if $s = \mathsf{B}_i$ **then**
      Blend new cell with cell i by using the special blending reactions only (see also Algorithm 16).
    end if
  end if
end while

6.4.6 aB System Formalization

Formally speaking an aB membrane system is a construct

$$\Pi = (V, S, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m), \tag{6.20}$$

where

1. V is an alphabet. Its elements are called *objects* or *molecules*;
2. S is an alphabet of special operator symbols with $S \cap V = \emptyset$;
3. μ is a membrane structure consisting of m membranes; all membranes (and hence the regions) are invectively labeled with $1, 2, \ldots, m$, where m is called the degree of Π;
4. $w_i, 1 \leq i \leq m$ are strings which represent multisets over V associated with the regions $1, 2, \ldots, m$ of μ;
5. Let $r = (u, s, v)$ be an *evolution rule* or *reaction*, usually written in the form $r = u \rightarrow sv$;
6. Let $b = (u, a, s, v)$ be a *blending evolution rule* or *blending reaction*, usually written in the form $r = u \rightarrow_a sv$, where a represents the activity of the reaction;
7. Let R^* be the set of all possible reactions r and b;
8. s is a special operator symbol from S, u, v are multisets over $V \cup R^*$;
9. $R_i, 1 \leq i \leq m$ are finite sets of *evolution rules* over $V \cup R^*$ associated with each membrane.

6.5 Wrap-Up

In this chapter, we have first seen a candidate model for membrane systems that are intended to be implemented on an amorphous computer. The so-called atP membrane system was basically inspired by tissue P systems, but was too limited.

In the next section, aP membrane systems were introduced. aP membrane systems form a special class of membrane systems that are inspired by Paun's classical membrane systems, by tissue P systems, and that were adapted for a future implementation on an amorphous computer substrate. The main difference to classical P systems is the execution policy of the reaction, which is kept completely stochastic and nondeterministic. Besides the increasing difficulty to "program" such a chemistry, there are also several advantages. The most important issue is probably that completely stochastic reaction dynamics are more robust — as they basically require to store the "information" in concentrations instead of single molecules — and are therefore better suited for amorphous computers.

A further powerful feature of aP membrane systems is the possibility to rewrite reactions, i.e., to consider reactions as molecules too. This allows to self-modify the chemistry and represents in some way also an amalgamation of program (i.e., reactions) and data (i.e., molecules).

In the last section, an extended class of aP membrane systems is presented. aB membrane systems unify the computational blending approach called C-Blending (as presented in Section 5.6) and aP membrane systems. A newly introduced special operator symbol allows to blend two existing cells together and to create a new one.

So far, membrane systems were mainly deterministic and entirely programmed. The new blending operator for a first time allows to introduce some "creativity" into the system. This makes of course little sense for systems which were created to compute the same function over and over with different arguments. The situation is naturally different for a robotic controller, for example, or an amorphous computer embedded in a paint which has to measure the temperature and to react to the environment accordingly.

In summary, the B blending operator becomes interesting when a system has to adapt to a changing environment and where a "continuous" operation is required.

Finally, I'd like to mention that I did *not* deal with the construction in general of membrane systems. This actually a very weak point of classical P systems as well. One always simply *assumes* that a certain membrane system exists. This does not cause only problem from the theoretical point of view, but in reality, the membranes and objects must first be created by a process which is usually all but trivial.

CHAPTER 7

Amorphons and the Circuit Amorphous Computer

> The atmosphere surrounding the problem is terrible. Dense clouds of language lie about a crucial point. It is almost impossible to get through to it.
>
> *Notes for Lectures on Private Experience from The Philosophical Review*
> Ludwig Wittgenstein, 1968

7.1 Introduction

The goal of this chapter is to present a candidate amorphous computer model and a candidate conceptual architecture of an *amorphon*, the basic elements of the amorphous computer. Please be aware that this chapter presents several ideas and concepts which I was unable to fully implement, simulate, and test.

Note that it was *not* the goal of this thesis to really implement an amorphous computer, as this would have been too complicated and it would have been impossible to build a sufficient number of elements anyway.

In the first part, the Circuit Amorphous Computer model is presented. Compared to the MIT model, my model has a modified communication structure.

In the second part of this chapter, a possible architecture for an abstract amorphon is presented.

In the last section, I will illustrate how hierarchical membrane structures might be created.

7.2 The Amorphous Computer Model Used

The amorphous computing concept and its paradigms have been presented in details in Section 2.9 already. In this section I will briefly present the modified amorphous computer model used henceforth and which will be at the basis of amorphous membrane computing.

7.2.1 Motivations

The MIT amorphous computer model application domain includes large scale distributed computers such as *paintables*, i.e., large numbers of particles embedded in a paint. This kind of particle distribution also asks for particular applications which are able to fully exploit the possible massive parallel sensor input (such distributions and applications are sometimes also known by the name *Smart Dust* [370,451]). The slightly modified amorphous computing paradigm presented in this section is somehow aimed at a different application domain and the main questions are also slightly different. Our original ideas were motivated by the following considerations (see also Chapter 1, especially Section 1.4.1):

- Abandon the world of highly regular, homogeneous, and perfect (theoretical and physical) structures, such as CAs, FPGAs, etc.
- Do not impose synchrony.
- Propose a computational model that allows to compute in novel, highly integrated (probably still electronic) substrates that possess a random interconnection structure and unreliable components. We believe that a random topology can potentially exploit new manufacturing technologies.
- Explore the characteristics and properties of such architectures and circuits.

The original amorphous computer communication model assumes that all processors have a circular broadcast of approximately the same fixed radius (large compared to the fixed size of a processor) and share a single channel. The main reason for this that the amorphous particles communicate by means of radio transmitters—all sending and receiving on the same channel—with their neighbors.

As the present work is aimed at (see also Section 1.4.1) novel computational substrates based on billions of randomly arranged and interconnected particles, we used an adapted and somehow restricted amorphous computer model. The basics remain the same, however, the only modifications are concerned with the communication and interconnection model which is modified as follows:

- Each particle i receives inputs from an average of K_{avg} of its neighbors that lie within an average radius of c_{avg}
- All communication channels are single and unidirectional channels.
- The particles are distributed on a two-dimensional surface.
- The particles as well as the communication channels are fixed, i.e., they do not move, but they might be unreliable.

An amorphous computer that respects the above modifications shall be called *Circuit Amorphous Computer (CAC)*.

Figure 7.1 shows a typical application: novel computational substrate based on billions of randomly arranged and interconnected particles. Besides nanotech, molecular electronics, and other alternative technologies, there is yet another pretty recent technology which begins to become the more and more relevant: printing techniques to create cheap and flexible sheets of transistors — a process that could radically change the way electronic circuits and for example flat-panel screens are being built.

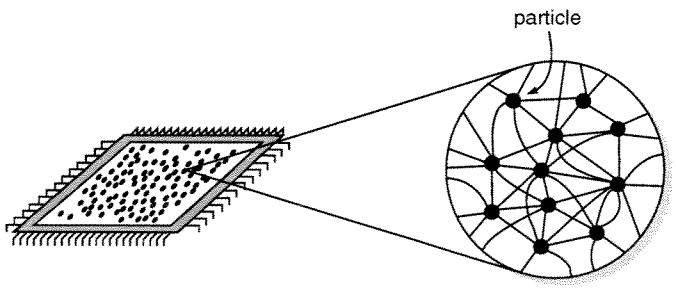

Figure 7.1: An application of the *Circuit Amorphous Computer (CAC)*: a novel integrated circuit (silicon or not) based on billions of randomly arranged and interconnected particles. The interconnections as well as the particles are unreliable.

7.2.2 Definitions and Formalizations

More formally speaking, a Circuit Amorphous Computer might be defined as in Definition 7.2.1. The definition is also illustrated in Figure 7.2.

Definition 7.2.1 (Circuit Amorphous Computer)
A Circuit Amorphous Computer can be defined as a triple

$$\mathsf{CAMC} = (V, E, K_{avg}, c_{avg}) \tag{7.1}$$

that represents a directed graph and where:

1. $V = \{(x_1, y_1, c_1), \ldots (x_N, y_N, c_N)\}$ *is the set of particles, i.e. the nodes of the graph,* (x_i, y_i) *is the random physical position of the particle* i *within the Circuit Amorphous Computer related to some coordinate system (see Figure 7.2),* $C_{min} < c_i < C_{max}$ *is the communication radius of particle* i*,* C_{min} *and* C_{max} *guarantee that* c_i *is of approximatively the same value for all nodes;*
2. $N = |V|$ *is the number of particles;*
3. $E = \{(i, j) | i, j \in V\}$ *is the set of directed edges;*
4. $n(i) = \{j | (i, j) \in E\}$ *is the local neighborhood of processor* i*;*
5. $d(i) = |n(i)|$ *is the degree of the neighborhood of processor* i*; and*
6. $K_{avg} = \frac{1}{N} \sum_{i=1}^{N} d(i) \ll N$ *is the average neighborhood size.*

■

It is assumed that the physical position as well as the interconnection topology are determined at the construction of the amorphous computer. Let us see how this could be done. Algorithm 18 allows to establish an interconnection topology that respects approximately the constraints of Definition 7.2.1.

Algorithm 18 Building a Circuit Amorphous Computer

1: Let $V = \{(x_1, y_1), \ldots (x_N, y_N)\}$ be a the set of particles, $N = |V|$.
2: Let K_{avg} be the average interconnectivity.
3: Let c_i be the communication radius of particle i.
4: **for** each particle i **do**
5: Randomly choose K_i from the interval $[K_{avg} - \delta, K_{avg} + \delta]$.
6: Randomly connect (as incoming connections) K_i neighbors that lie within a radius of c_i.
7: **end for**

The following example shows how top create an Circuit Amorphous Computer with the MATLAB toolbox:

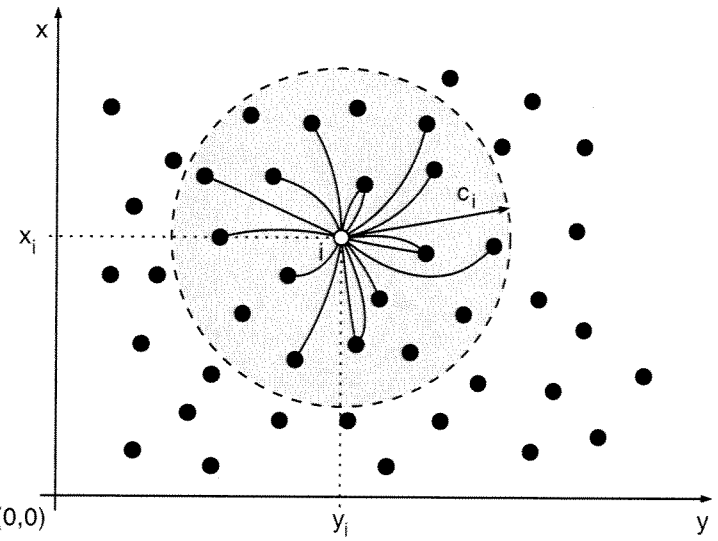

Figure 7.2: Illustration of the Circuit Amorphous Computer communication model. Each particle i receives inputs from K neighbors within a radius of c_{avg} on the average. Multiple connections to the same neighbors are allowed.

MATLAB AMB toolbox example:

```
>> [nodes edges] = createCAC(10000, [10 10], 2, 1);
>> displayCAC(nodes)
```

First, an Circuit Amorphous Computer ($N = 1000$ nodes, max. coordinates: $(10, 10)$, $k_{avg} = 2$, $c_{avg} = 1$) is created according to Algorithm 18. The coordinates of the nodes are stored in `nodes`, the edges in `edges`. The particle distribution can then be visualized by `displayCAC`. The result is shown in Figure 7.3

The function `displayCACGraph` further allows to plot the amorphons (nodes) and their interconnections (edges) in the form of a graph. The following MATLAB example results in Figure 7.4.

MATLAB AMB toolbox example:

```
>> [nodes edges] = createCAC(200, [10 10], 10, 2);
>> displayCACGraph(nodes, edges);
```

In the following, I will use as an illustrative example the Circuit Amorphous Computer as depicted in Figure 7.5.

Finally, it is worth mentioning that a Circuit Amorphous Computer has some similarities with the BLOB computing project (see Section 2.9.5),

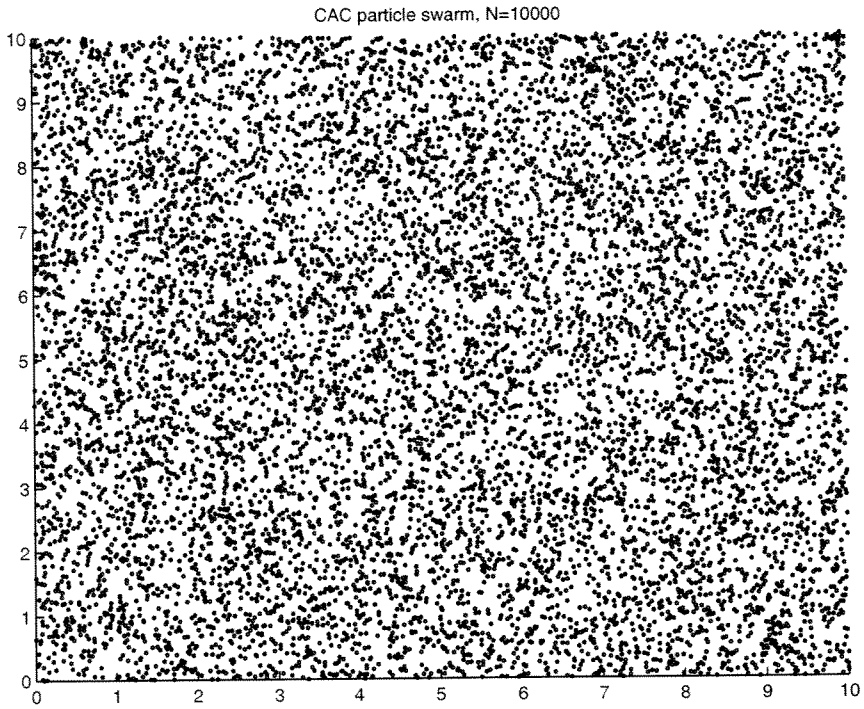

Figure 7.3: Particle swarm of a Circuit Amorphous Computer ($N = 10'000$ particles).

which is also based on a asynchronously operating network of automata with non-deterministic evolution rules. The network is defined as a non-oriented graph with an arbitrary architecture. Gruau *et al.* [173] also defined a *nearest neighborhood* $K(a)$ of an automaton a that specifies the automata connected to a. States are considered as colors in blobs and the evolution rules specify how the colors change.

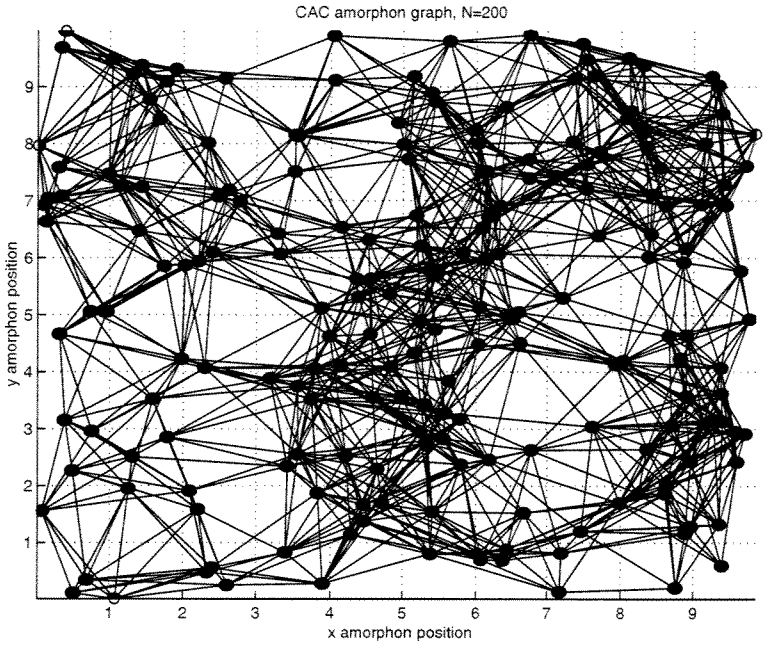

Figure 7.4: Graph of a Circuit Amorphous Computer ($N = 200$ amorphons, $K_{avg} = 10$, $r = 2$).

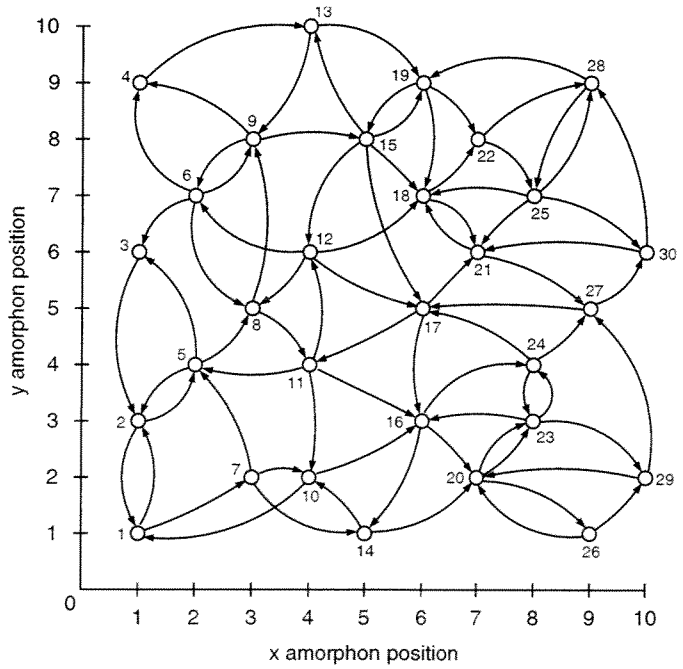

Figure 7.5: A Circuit Amorphous Computer example as it will be used for explanations.

7.3 An Abstract Amorphon

In a first step, the Circuit Amorphous Computer amorphon model shall be considered as an abstract black box that provides a set of primitives and that reacts to the environment. The amorphon has an internal state and several internal variables store relevant values such as gradients.

The amorphon's operation and execution model is rather simple and kept hardware-friendly. Figure 7.6 illustrates the basis of the message queue and message execution model of an abstract amorphon. All incoming and accepted messages an amorphon receives are inserted into the message queue `mqueue` where they are processed sequentially by the amorphon's processing unit. The `reject`-list contains a list of message types that will be rejected by the amorphon. These messages will be sent back to a neighbor chosen at random. This basically allows to implement membranes that can only be penetrated by certain chemicals.

Each amorphon possesses a locally operating clock that may be different from the clocks of his neighbors. The clock will basically be used for the chemcial reactor, otherwise all communications and operations are kept asynchronous. As we will see later, the communications are based on data packets will will find their way on their own through the network of interconnections.

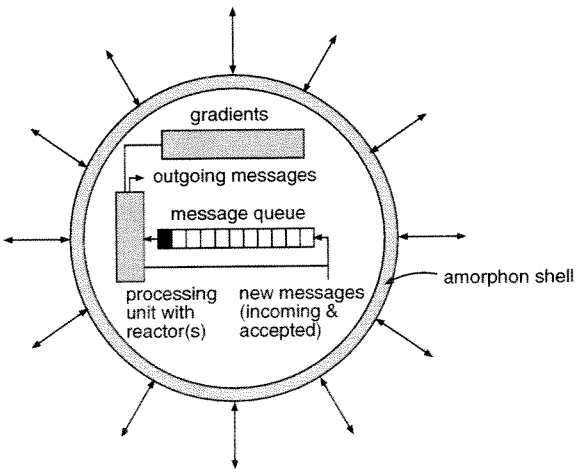

Figure 7.6: Illustration of the message queue and message execution model of an abstract amorphon. Messages not allowed to enter the amorphon message queue (blocked by the "shell") are sent back to a neighbor chosen at random. The execution unit also contains the reactor for the artificial chemistry. A special module handles gradient values.

The following list summarizes the principal characteristics of an abstract

amorphon:

- The amorphon can only access the first message of the message queue. All messages received are executed sequentially.
- Each amorphon can access the public variables of its immediate neighbors as specified in the `neighbors` variable. The number of neighbors is known, but not their physical position.
- There is no global clock that synchronizes the different amorphons.
- Each amorphon is initialized with the same program. This might typically be done at fabrication time.
- Each amorphon has limited capacity only (this is actually a parameter which can be changed easily).
- Everything should be based on the chemical metaphor as illustrated on page 255. This means that whenever possible, one should used molecules and reactions.

7.3.1 Internal Variables

Table 7.1 summarizes the internal variables and data structures as used in the MATLAB simulation.

In addition, Table 7.2 provides the list of cell states an amorphon can take. This is also illustrated in Figure 7.7. It should be mentioned that some of these variables might become unnecessary for a hardware implementation and that I did not use all states.

7.3.2 A Distributed Reactor Network

Remember from Chapter 6 that the reaction dynamics of each cellular membrane of an aP and aB membrane system are modeled by a stochastic reactor. The idea behind this was to be able to later use a distributed reactor network.

The execution unit as shown in Figure 7.6 also contains the reactor which will me responsible to develop the artificial chemistry of the membrane systems. Naturally, such a reactor has a physically limited capacity, which I wanted to include in the model as it would not have been very realistic to suppose amorphons with unlimited resources.

As illustrated in Figure 7.8, each amorphon can take part in a distributed reaction network of well-stirred reactors. As each reactor has a limited capacity only, i.e., he can only contain a limited number of reactions and molecules, the entire networks will also have a limited capacity, however, the distributed reactor network can be extended by simply adding new reactors.

Variable	Description
Public:	
`state`	Amorphon state (see Table 7.2)
`mqueue`	Message queue
`gradients`	List of gradients (names and values)
`neighbors`	List of neighbors
`reject`	List of message types to reject
`resources`	Indicates the resources used (0-100)
Private:	
`id`	Random amorphon identification number
`cellid`	Identification number of the cell the amorphon is part of
`color`	Amorphon color (used for displaying)
`position`	Amorphon position (x,y)

Table 7.1: Summary of the internal variables of an abstract amorphon as used in simulation. Public variables are accessible to any amorphon resource, private variables are for internal use in relation to the simulation only.

State	Description
`cellEmpty`	Unused amorphon
`cellBorder`	Amorphon is part of the cellular membrane
`cellMatter`	Cell matter (neither border nor core)
`cellCore`	Cellular core (initiated the cellular growth)

Table 7.2: Amorphon states

Note that the interconnections between the reactors basically correspond to the interconnections provided by the amorphous computer.

All rules associated to a cell are distributed to all reactors within the reaction network. Rules can also change during a computation and will therefore have to be re-distributed to all reactors within a cell. One could however also imagine that both rules and molecules randomly move around in the distributed reaction network.

In my concept, however, each molecule has a certain probability p_l to leave a reactor and to move to one of the neighboring reactors. The higher p_l, the better the entire reaction network is "stirred", but the more traffic occurs in the network, which might become a problematic issue if bandwidth is limited.

Each amorphon reactor basically operates at its own speed and there is no global clock signal or whatsoever available. This is of course a major difference to classical P systems where the

- rules are applied in a maximum parallel manner, i.e., all rules that can

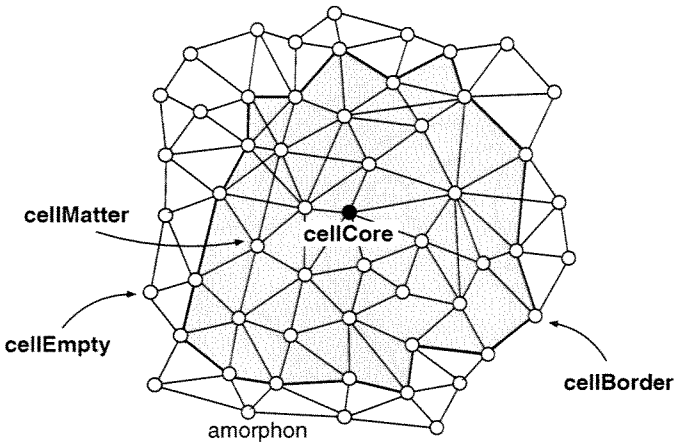

Figure 7.7: Illustration of the different states an amorphon can take. The implementation of the membranes will not always make use of all of these states. The special role of the `cellCore` is emit a gradient that allows to the other cells to identify the cell and to find its origin.

be applied have to be applied by non-deterministically choosing rules till no further rule can be applied to the remaining objects;

- rules can be applied in the same step as many times as we want; and
- rules are applied synchronously within each membrane.

The problem with these restrictions is — as we have already seen in Section 4.8 — that a physical implementation in hardware is all but straightforward. Remember that there is no global clock signal in an amorphous computer and that it would not be trivial anyway to distribute a synchronous clock signal on a large surface. Note that his was actually also the main design challenge while we built the BioWall (Section 4.3): distributing the global clock signal on a surface of $1m \times 6m$.

7.3.3 Communication Model and Primitives

The communication model with its primitives is based on sending and exchanging messages between the amorphons. In our case, the message contains all necessary information to find its destination. Therefore, no global routing tables or whatsoever are necessary. The message is either a broadcast or follows a gradient. This bears the enormous advantage that failures in connections — at least as long as the amorphous graph is still fully connected — do not affect the global systems behavior since the message will find another way to their destination.

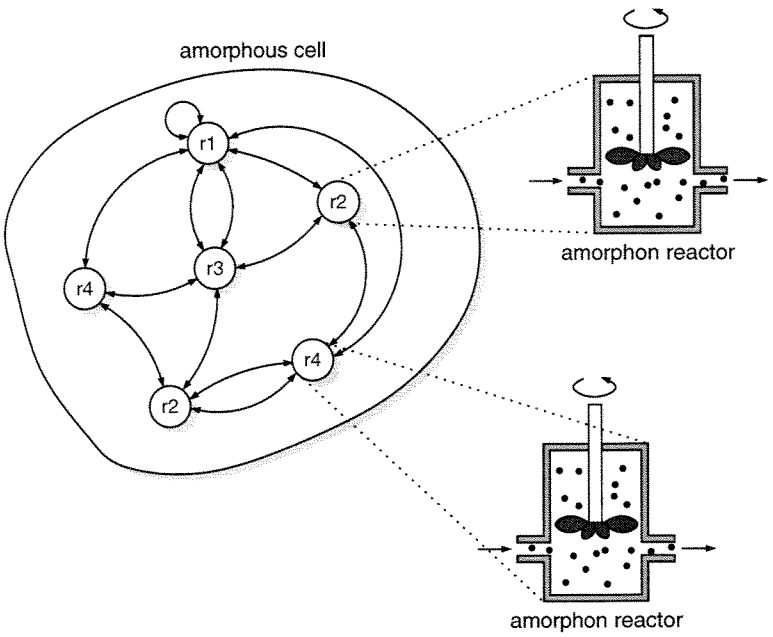

Figure 7.8: A distributed network of "well-stirred" reactors.

The standard message is composed of a header and of a data part as illustrated in Figure 7.9. The header contains all necessary information to decide whether a message should be further analyzed or sent further. In case of further analysis, the message will always be inserted into the amorphon's message queue, otherwise it will be sent to one of the immediate neighbors chosen at random. The data part of a message can of course itself contain a header again, not unlike the well-known ISO communication layer model [401], which is, despite its age and rigidity, still widely used.

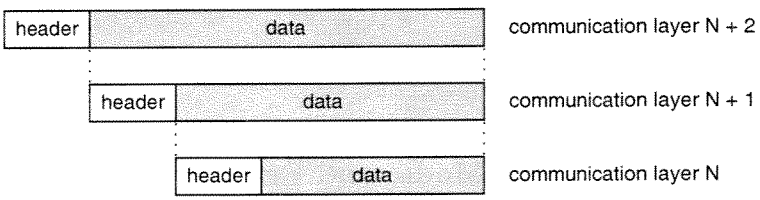

Figure 7.9: An amorphon message is composed of a header and of data part. The data part can itself contain a header and a data part, etc., which allows to have different levels of communication and abstraction.

There exist of course almost uncountable ways to communicate between a sender and a receiver. We are however looking for a simple, hardware-friendly mean that which is based on the chemical paradigms and which takes into account the special structure of the underlying amorphous computer.

We have therefore opted for the following very few communication primitives on our Circuit Amorphous Computer:

- **Local Broadcast:** The amorphon sends the message M to all of its immediate (i.e., local) neighbors. The broadcast can be limited to a certain number of maximum hops ($maxHops$) or to amorphons having a certain state ($stopState$).
- **Follow Gradient:** The message M follows gradient G upwards (i.e., towards increasing gradient values) to its source (gradients will be further described in Section 7.3.4).
- **Search:** The message M will randomly move around until it has found an amorphon that satisfies the specified state ($searchState$) or until it has executed a maximum number of hops ($maxHops$).

The primitives are summarized in Table 7.3.

Syntax	Description
broadcast($stopState$, $maxHops$, M)	Broadcast message M to immediate neighbors.
gradient(G, M)	Deliver message M to the source of gradient G.
search($searchState$, $maxHops$, M)	Search an amorphon with a certain state.

Table 7.3: Syntax of communication primitives

7.3.4 Gradients

Chemical *gradients* are absolutely crucial in any biological system, especially also in developmental biology (see for example [466]). Many biological "components" secrete chemicals that are then diffused throughout the surrounding environment. When talking about gradients, one usually refers to a gradient of a chemical concentration that decreases in its value when moving away from its source. The *chemical concentration* therefore provides an estimate of distance from the source of the chemical. On the other hand, a *tropism*[1] allows to sense the direction of the chemical gradient by comparing neighboring chemical values.

[1] **tropism**, involuntary response of an organism, or part of an organism, involving orientation toward (positive tropism) or away from (negative tropism) one or more external stimuli. The term tropism is usually applied to growth and turgor movements in plants. Source: `http://www.infoplease.com`.

The amorphon computing project (see Section 2.9) also extensively makes use of gradients. For more information see for example [47, 80, 281, 283].

In our system, we make use of artificial gradients and tropisms as biologically-inspired paradigms to guide particles, either to find a certain point of origin (e.g., the cellular core) or the spread away from a certain point. Gradients and tropisms are robust primitives that are insensitive to failures of individual components, insensitive to the placement of the components (i.e., amorphons), and trade off precision for reliability.

Unlike in Nagpal's thesis [283], our gradients decrease as one moves away from the source. This implies that the initial gradient value has to be initialized by some maximum value.

More formally speaking, a gradient is a couple

$$g = (name, value) \tag{7.2}$$

where *name* is the gradient name (i.e., the type of chemical) and *value* represents the *concentration* of the chemical. Each amorphon stores the concentrations of the gradients it receives and can create gradients by sending a message, e.g., a couple $g = (name, value)$, to its immediate neighbors. The neighbors then decrement the value by one, each stores the maximal value it has heard for any particular gradient name. Nagpal further averages the gradient value over their neighbors to improve the distance estimate.

Algorithm 19 presents an method to continuously update and spread gradients throughout the network. All amorphons execute this algorithm in parallel and the gradients then propagate over the entire Circuit Amorphous Computer. Figure 7.10 shows an example of a gradient that propagates throughout a graph from the origin `cellCore` having an initial gradient concentration of 10.

Figure 7.11 shows a typical gradient surface. The origin of the gradient is located in $(4.5, 5.5)$. The surface is interpolated over the irregular arrangement of $N = 100$ amorphous particles. The interpolation does not accurately represent the original gradients of the Circuit Amorphous Computer and one might for example see certain local maxima. Algorithm 19 ensures, however, that there is always a direct interconnection from a given node with a certain chemical concentration to a neighboring node with a higher concentration. When no neighboring node with a higher concentration exists, then the node must be the origin of the gradient.

A moving message can easily find the origin of a chemical gradient by always moving into the direction of a higher gradient concentration. As already said above, when it arrives at a point where no neighbor with a higher concentration exists, the source has been reached.

The MATLAB code to generate similar figures to Figure 7.11 is as follows:

Algorithm 19 Update and Propagate Gradient Concentrations

Require: $G = \{g_1 = (n_1, v_1), \ldots, g_N = (n_N, v_N)\}$
Require: $Q = \{gq_1 = (nq_1, vq_1), \ldots, gq_M = (nq_M, vq_M)\}$
1: // G = set of already existing gradients
2: // Q = set of incoming gradient messages
3: **loop**
4: **for all** gradients $gq_i \in Q$ **do**
5: **if** $\exists g_j \in G | gq_i(nq_i) = g_j(n_j)$ **then**
6: **if** $(gq_i(vq_i) - 1) > g_j(v_i)$ **then**
7: // Update gradient value in G
8: $g_j = (n_j, vq_i - 1)$
9: **end if**
10: **else**
11: // Add non-existing gradient
12: $G = G \cup \{g_{N+1} = (nq_i, vq_i - 1)\}$
13: **end if**
14: Send gq_j to all immediate neighbors
15: // Remove gradient from incoming set
16: $Q = Q \setminus gq_j$
17: **end for**
18: **end loop**

MATLAB AMB toolbox example:

```
>> maxpos = [10 10];
>> N = 100;
>> [coord edges] = createCAC(N, maxpos, 10, 2);
>> a = initAmorphonEnsemble(N,coord,edges);
>> a(50).mqueue = initAmorphonMessage({'initCell'},...
                                        [1 0 0],[]);
>> a = runAmorphonEnsemble(a);
>> plotGradient(a,'CC1', maxpos);
```

As an illustration Figure 7.12 shows that path of a chemcial taken to find the `cellCore` located at amorphon number 17. It's gradient value is 100. Each message also counts is number of *hops* and it is for example stuck in a dead-end, it will randomly choose a certain path after a maximum count of hops has been reached. This is absolutely necessary as there might not always exist a path in the direction of a increasing gradient value.

The following MATLAB command allows to test the *gradient* command and to search for the `cellCore`:

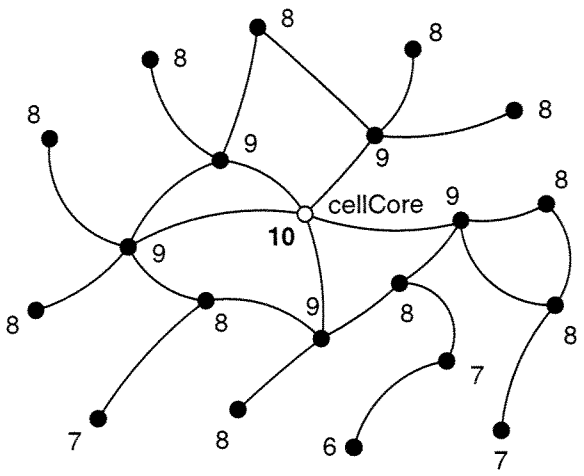

Figure 7.10: A gradient spreads throughout the graph from the origin `cellCore` with an initial gradient concentration of 10.

MATLAB AMB toolbox example:

```
>> testSearchCellCore(40)
```

Gradient-based communication of course also comes with several drawbacks. The most important one is that communication delays and routes are not predictable and that the time necessary to send a message to a certain destination is highly dependent on the environment, the number of connections, and the number of errors in the network. This makes the entire system somehow unpredictable, but this is the price to pay for robustness.

7.3.5 A Note on Redundancy and Fault-Tolerance

The distributed reaction network provides a highly robust infrastructure in addition to the potential robustness and redundancy of artificial chemistries (if correctly used). As we have seen in Chapter 6 already, one can for example encode the state of a state machine in the concentration of a certain chemical. If the concentration is above a certain level, the state will be considered as valid, otherwise as invalid. Obviously, the loss of chemicals will not globally affect the state of the machine as long as the concentration does not fall below the limit.

The reactor network can therefore even be leaky. And in the case of extreme losses of chemicals, one might provide special reactions which regenerate the molecules and hold the concentration constant. This is actually really an interesting characteristic of artificial chemistries. If the information is encoded in a smart way by means of the chemicals, then fault-tolerance

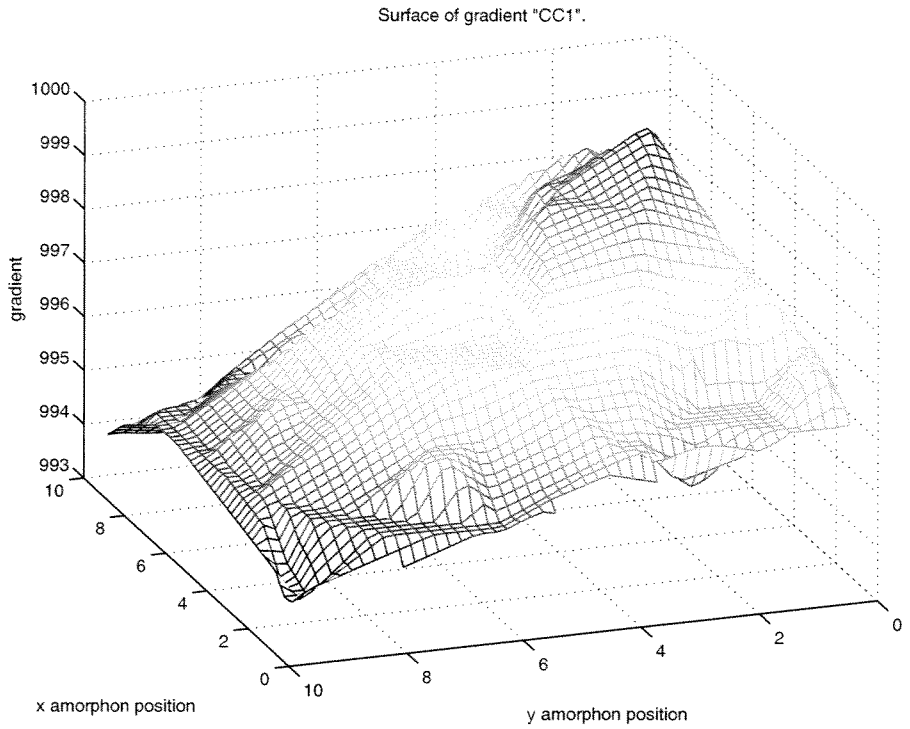

Figure 7.11: Illustration of a gradient surface. The gradient "CC1" (`coreCell` gradient) is displayed. The `coreCell` is located at position $(4.4577, 5.4667)$. The maximum gradient value at the origin is 1000. The Circuit Amorphous Computer used is built up by $N = 100$ amorphons, has an interconnectivity of $K_{avg} = 10$ incoming links per node, and the amorphons communicate within a radius of $r = 2$. The gradient surface is interpolated over the irregular amorphous computer grid.

comes almost for free!

In addition, the Circuit Amorphous Computer provides a highly redundant communication structure (depending on the parameters), as there are usually several connections available between neighboring amorphons. Since the message routing is based on chemical gradients, a faulty connection will not at all affect the global behavior.

Gradients are of course also themselves highly robust to perturbations and faults. As their chemicals constantly spread throughout the amorphon network, a faulty amorphon or connection will be instantly bypassed by messages that follow a gradient concentration.

So far we have only discussed the redundancy on the level of the membrane system's artificial chemistry, however, we should also consider what will happen when an entire amorphon fails (and not only a reactor within

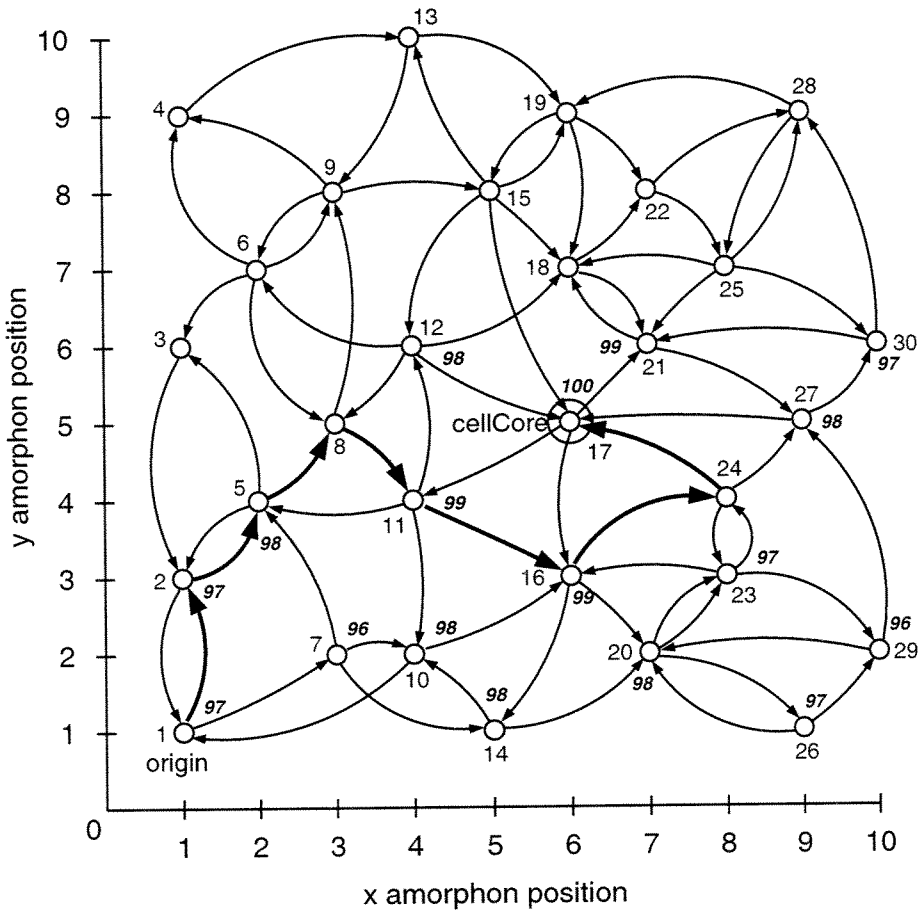

Figure 7.12: The path (bold arrows) of a message taken which searches the `cellCore` located at amorphon 17. The small numbers in bold italics represent the gradient values. The `cellCore` has a value of 100.

an amorphon).

The loss of chemicals and reactions might be compensated by the other reactors in case of a smart and redundant information encoding (see above and also Section 6.3.5), however, there are various other data stored in the amorphons. For example, if the internal state is lost, an amorphon and its neighbors will no longer know whether it is part of a cell or not.

The most straightforward solution for such a failure is the following: each amorphon stores the information of all of its neighbors too. This of course means that the information has to be regularly updated by means of a local broadcast. Each amorphon sends its contents (preferably in a compressed form) to all its local neighbors. On the other hand, each amorphon monitors its immediate neighbors and will either be able to restore lost information (in

case of a transitory failure) or to "replace" the faulty amorphon by sending the contents to an empty amorphon in the neighborhood.

Compared to the Embryonics project (see Section 4.2), where each cell stored the entire program of the machine, the local redundancy comes at a much lower price in terms of resources required.

7.4 Creating Membrane Structures

One can say that creating hierarchical membrane structures on the Circuit Amorphous Computer represents a key issue of implementing aP and aB membrane systems. The other operations are basically chemical based only and can be executed in the reaction network.

My first idea for creating the membranes was to directly map the membrane structure onto the amorphous computer. This is illustrated in Figure 7.13. This approach bears however several drawbacks:

- If the membrane creation process starts with the skin membrane first, a special growth mechanism would be necessary to "push" to enlarge the skin membrane each time a new inner membrane is being created.
- When one of the inner cells requires more amorphons because the molecules and reactions do no longer fit into the allocated reactors, each upper-level cell would need to expanded towards the exterior, which of course would take some time.

On the other hand, the method proposed allows to gradually and "trouble-free" create membranes on Circuit Amorphous Computer if the approximate size of the cells is know. In a first step, the skin region will be created and enlarged until all inner cells will fit into the membrane.

Algorithms 20 - 23 allow to create a single cell by means of two commands only: `initCell`, `enlargeCell`. At initialization, each amorphon is set to an initial state by means of the `initAmorphon` command, where as the `setBorder` command is used by `enlargeCell` to push the cellular membrane further.

Algorithm 20 Amorphon Program: `initAmorphon`

```
1: amorphonID = random number.
2: cellID = 0.
3: cellState = cellEmpty.
4: messageQueue = empty.
5: gradients = empty.
```

Algorithm 21 Amorphon Program: `initCell`

```
1: loop
2:   if messageQueue = initCell then
3:     Broadcast (setBorder, cellID) to all immediate neighboring amor-
       phon.
4:     cellState = cellCore.
5:     cellID = amorphonID.
6:   end if
7: end loop
```

The creation of the single cell membrane is illustrated in Figures 7.14 and 7.15.

The creation of a cell is initialized by sending the `initCell` command to an amorphon's message queue. This can be done by the following MATLAB commands:

MATLAB AMB toolbox example:

```
>> [coord edges] = createCAC(100, [10 10], 10, 2);
>> a = initAmorphonEnsemble(100,coord,edges);
>> [anb am] = amorphonEnsCenter(a,[10 10]);
>> a(anb).mqueue = initAmorphonMessage({'initCell',1});
>> a = runAmorphonEnsemble(a,100);
>> [coord color] = extractDisplayInformation(a);
>> displayCAC(coord, color);
```

First, a Circuit Amorphous Computer with $N = 100$ amorphons is created, then `morphonEnsCenter` determines the "geographical" center of the amorphon ensemble where the `initCell` message will be inserted into the message queue. The amorphon ensemble will then be simulated for 100 time steps in order to allow the gradients to distributed. Thereby, the cellular `cellBorder` will be spread from the `cellCore`to its immediate neighbors. This is illustrated in Figure 7.14.

Finally, by sending an `enlargeCell` command to one of the amorphons within the cell, the cellular membrane (`cellBorder`) will be enlarged by one amorphon. This mechanisms therefore allows adapt the cell to his growing needs. This is how `enlargeCell` is sent to an amorphon which is part of the current cell. In the MATLAB code below, the command is sent to the cellular core. Figure 7.15 illustrates how the cellular membrane has been enlarged.

Algorithm 22 Amorphon Program: `enlargeCell`

```
1: loop
2:   if messageQueue = enlargeCell then
3:     if cellState = cellBorder then
4:       Randomly choose an empty neighboring amorphon that is not
         part of the current cell.
5:       if no such amorphon exists then
6:         Send enlargeCell to a randomly chosen neighboring amor-
           phon that lies within the current cell.
7:       else
8:         Send (setBorder, cellID) to the new amorphon.
9:         cellState = cellMatter.
10:      end if
11:    else
12:      Send enlargeCell to a randomly chosen neighboring amorphon
         that lies within the current cell.
13:    end if
14:  end if
15: end loop
```

Algorithm 23 Amorphon Program: `setBorder`

```
1: loop
2:   if MessageQueue = (setBorder, cellID) then
3:     cellState = cellBorder.
4:     cellID = cellID.
5:   end if
6: end loop
```

MATLAB AMB toolbox example:

```
>> a(anb).mqueue = initAmorphonMessage({'enlargeCell',[]});
>> a = runAmorphonEnsemble(a,100);
```

Finally, in order to avoid the drawbacks of the above described approach, I also tried another method. Instead of directly mapping the membrane structure onto the Circuit Amorphous Computer, the cellular hierarchy might be flattened and their hierarchical structure kept by means of the relations (i.e., the communication channels) between the membranes. This was actually already the trick used for the first hardware implementation of P systems as presented in Section 4.8.

Figure 7.16 illustrates the implementation of three hierarchically ar-

ranged cells on a Circuit Amorphous Computer. The position of each cell in the hierarchy is stored within each cell, i.e., cell 2 knows that its upper-immediate cell is cell number 1. The communication between the cells is again purely gradient based. Cell 2 can send objects to cell 1 by means of the gradient which is constantly issued by the `cellCore` of cell 1.

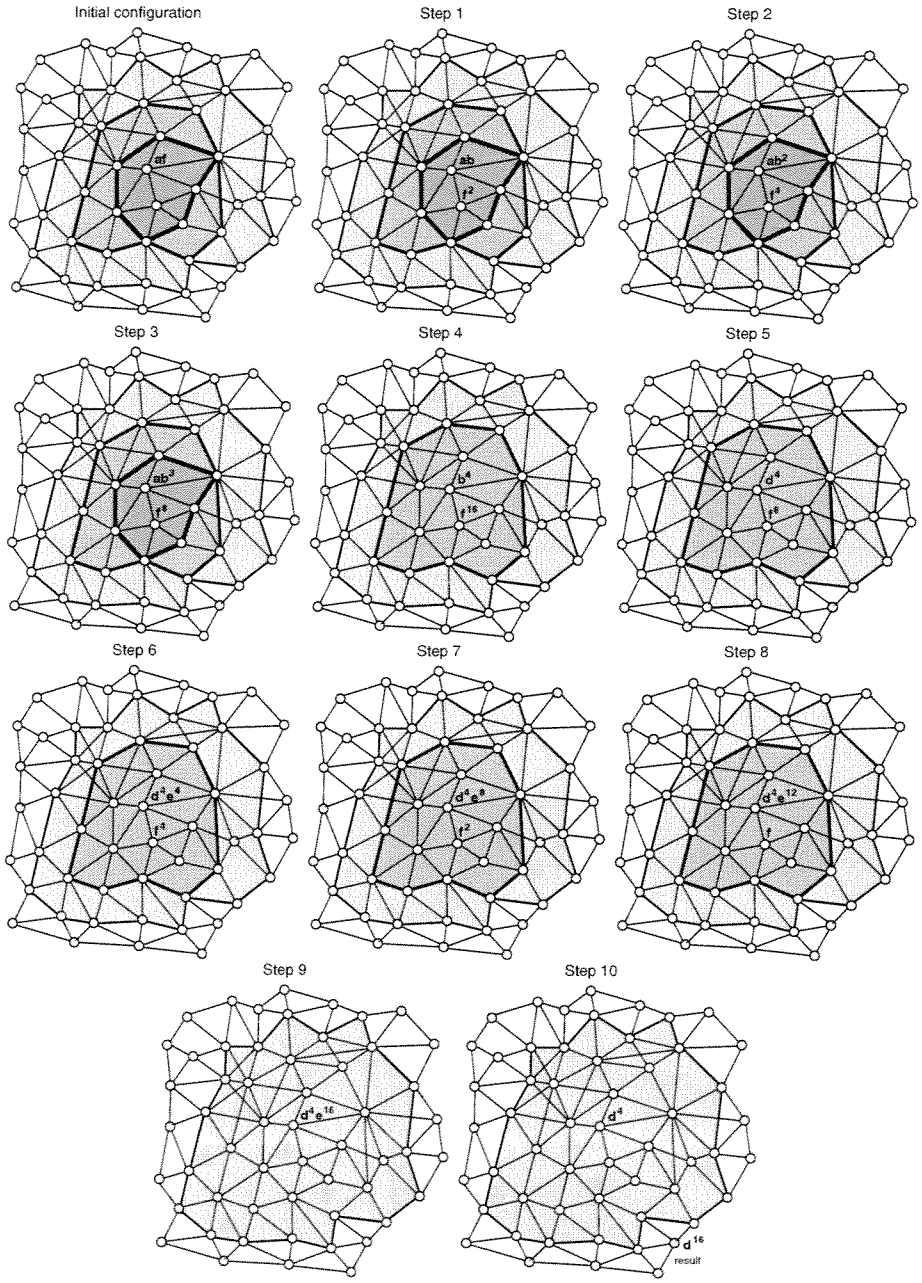

Figure 7.13: A halting computation in the amorphous membrane system that computes $\mathsf{Ps}(\Pi) = \{(n^2) | n \geq 1\}$. The rules are not displayed in the above computation.

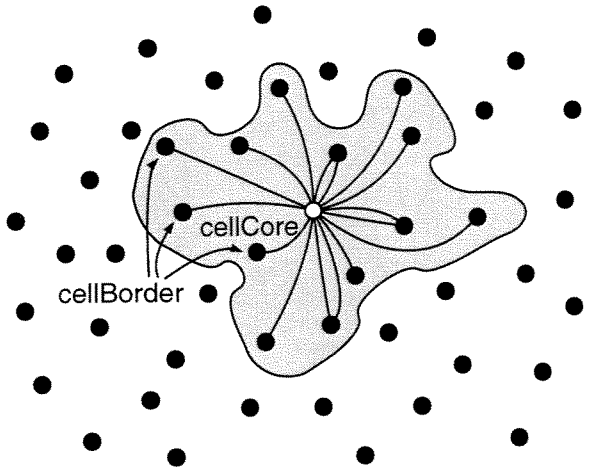

Figure 7.14: The first step of a cell creation consists in spreading the cellular border (`cellBorder`) from the `cellCore` to its immediate neighbors. The physical size of the cell, i.e., the number of amorphons used, depends on the neighborhood of the `cellCore`.

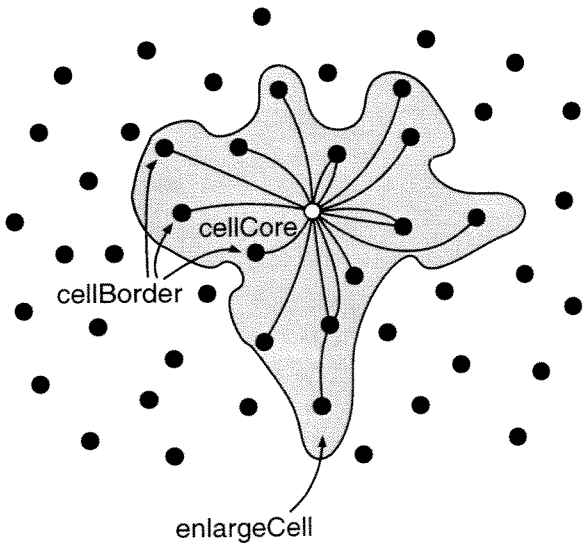

Figure 7.15: The cell can be enlarged by sending a `enlargeCell` command to one of the cell's amorphons.

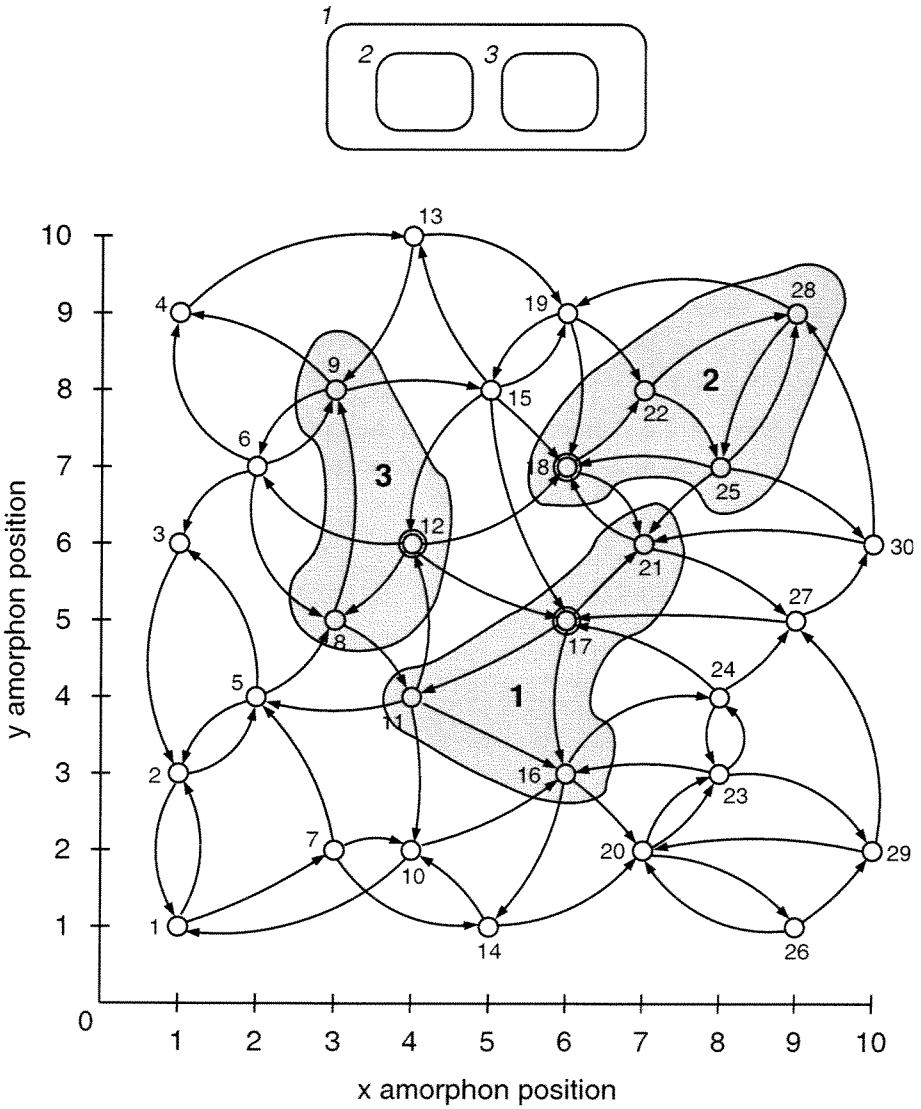

Figure 7.16: Implementation of cells on a Circuit Amorphous Computer.

7.5 Wrap-Up

In this chapter, the Circuit Amorphous Computer computer with its modified communication structure has been proposed. This special amorphous computer is intended to be used on a two-dimensional surface, where each particle i (i.e., amorphon) receives inputs from an average of K_{avg} of its neighbors that lie within an average radius of c_{avg}. Another important difference to the MIT amorphous model is that the particles as well as the communication channels are fixed, i.e., they do not move, but they might be unreliable.

Section 7.3 presented an abstract amorphon model as it might be used to implement membrane systems. Each amorphon contains a stochastic reactor with limited capacity. In order to be able to simulate membrane systems where the reactions and molecules do not fit into one reactor, the amorphons can form distributed reaction networks. All communications between the amorphons are guided by gradients only.

Finally, Section 7.4 delineates an approach on how to create membrane systems on the amorphon-based Circuit Amorphous Computer.

CHAPTER 8

Epilogue

> En toute chose il faut considérer la fin.
>
> *Fables, III, 5, Le Renard et la boue*
> Jean de la Fontaine

> The system is optimal, we just don't know what for.
>
> Gerald Jay Sussman

This thesis investigated a unique computational and organizational approach called *Amorphous Membrane Blending*. The approach mainly consisted in unifying three concepts, namely amorphous computing, membrane computing, and blending.

I will now draw some general conclusions and discuss some possible paths for future work on the basis on the present thesis.

For a list of the main contributions made in this thesis, see Section 1.4.3.

8.1 General Conclusions

Let us first recall the rather "philosophical" goal of this work as stated in the introduction:

> The present work is nothing else than a humble attempt to seek for further progress in computer science. The goal was not to investigate in an already existing research direction — such as neural networks, etc — but to try something new and "exciting".

This is exactly what I have done by trying to unify three separate research domains, namely (1) amorphous computing, (2) membrane computing, (3) and blending. The three topics share many common elements, but are sufficiently different to allow people to consider crazy what I have investigated.

From my point of view, the proposed concepts harbor a lot of potential, which I was unable yet to fully exploit for various reasons (see also below). The directions I have taken were fairly risky and it was totally impossible to anticipate the results, but that was part of the "deal".

When I have to explain to someone what I investigated in my thesis, the most usual question is "But what is all this good for?". This is of course a totally legitimate question and here are some answers:

- My goal was to explore a new direction in research which departs from what has been done so far.
- My goal was to investigate into an unknown direction to see whether the path is worth to be taken and to be continued.

I think that this is reason enough to explore the direction I have chosen.

Let us now briefly go through the work again and I will emphasize several points which seem important to me. Since all chapters already have a "wrap-up" section at the end, I will only mention the principal points here.

Chapter 2 first provided an introduction to "classical" and fairly well established biologically-inspired machines. When I started to look at bio-inspired approaches some years ago for the first time, I was not yet aware that most approaches are basically based on the very same ingredients and usually only differ in some small details. There are little investigations which radically depart from existing paradigms. This becomes maybe more clear when reading Section 2.8 (Towards POEtic Machines). Almost all models present are just variants of existing models.

Besides the more or less classical part of this chapter on POEtic machines and all their various derivates, Chapter 2 also introduced amorphous computers, membrane computing, and artificial chemistries, which are representatives of unconventional computing machines.

Chapter 3 was intended to motivate and introduce cellular machines and simple random (boolean) networks and to pave the way to irregular computational machines. In addition, this chapter was also intended to

provide a theoretical basis for the Circuit Amorphous Computer as presented in the last chapter (which is based on a special random graph). See also Section 8.2.2 for more information on future work on random structure.

The biologically-inspired hardware chapter basically only mentions classical, completely regular and cellular hardware machines. The BioWall cellular automata machine was for example presented in detail as well as other interesting and promising machines. A less conventional implementation is also presented in this chapter: a reconfigurable implementation of classical P systems, which also supports priorities and membrane creation and dissolution. This seems to be the first hardware implementation of membrane systems at all. The implementation made also fairly clear that classical membrane systems are not at all intended for any real-world hardware implementation and that some assumptions (i.e., the global synchronization signal, for example) might compromise the exploitation of the parallelism inherent in the membrane's chemistries.

When I first read Fauconnier and Turner's book [130], the idea of applying blending to membrane systems came into my mind, probably because all the illustrations directly suggested to used membrane systems. Although they claim in their book that blending cannot be modeled by a computer (without providing a clear reason, however), a few researchers are all the same working on computational blending. Chapter 5 therefore also includes a field review on computational blending algorithms.

The C-Blending approach represents an novel computational blending method intended for membrane systems and artificial chemistries. C-Blending is one possible approach to the creation of new chemistries, but many other algorithm might be imagined. This difficulty is probably best compared with the large number of methods that exist in the domain of evolutionary algorithms. Depending on the problem, the encoding, the population size, etc., a different crossover and mutation operator will be chosen. The general framework remains the same, but the details change.

In blending, there might be artificial chemistries where one method will work well, but not another. The general problem solver that provides optimal methods and solutions to all kinds of problems does unfortunately not (yet?) exist.

The next step consisted in proposed special membrane systems that were

- optimized for an implementation on an amorphous computer, and
- that would allow to implement the C-Blending approach.

The main systems proposed are the aP and the aB membrane systems. aP membrane systems form a special class of membrane systems that are inspired by Paun's classical membrane systems, by tissue P systems, and

that were adapted for a future implementation on an amorphous computer substrate. The main difference to classical P systems is the execution policy of the reaction, which is kept completely stochastic and nondeterministic. Besides the increasing difficulty to "program" such a chemistry, there are also several advantages: the most important issue is probably that completely stochastic reaction dynamics are more robust — as they basically require to store the "information" in concentrations instead of single molecules — and are therefore better suited for amorphous computers.

A further powerful feature of aP membrane systems is the possibility to rewrite reactions, i.e., to consider reactions as molecules too. This allows to self-modify the chemistry and represents in some way also an amalgamation of program (i.e., reactions) and data (i.e., molecules).

On the other hand, aB membrane systems unify the computational blending approach called C-Blending and aP membrane systems. A newly introduced special operator symbol allows to blend two existing cells together and to create a new one. So far, membrane systems were mainly deterministic and entirely programmed. The new blending operator for a first time allows to introduce some "creativity" into the system. This makes of course little sense for systems that were created to compute the same function over and over with different arguments. The situation is naturally different for a robotic controller, for example, or an amorphous computer embedded in a paint that has to measure the temperature and to react to the environment accordingly in some way. The B blending operator therefore becomes interesting only when a systems has to adapt to a changing environment and where a "continuous" operation is required.

In order to implement membrane systems combined with computational blending on amorphous computers, the *Circuit Amorphous Computer* was proposed in the last chapter together with the conceptual architecture of its basic building element, the *Amorphon*.

Throughout the work I tried to emphasize the various properties which resulted from the unification of the different concepts. For example, the cellular structures allow to create dynamical hierarchies and growing systems whereas the artificial chemistries represent an ideal mean to compute on the uncertain and irregular substrate of an amorphous computer. Finally, the computational blending proposed presents an inventive method to create and modify membrane systems dynamically.

The characteristics and limits of the concepts proposed were analyzed and validated using various examples and small applications.

Finally, and for the sake of completeness, here comes a short and certainly incomplete list of some general problems and critics of this thesis:

- I was unable to simulate complete systems using the MATLAB simu-

lator I developed. The computational limitations became very serious when trying to simulate several hundreds of amorphons.

- I was unable to provide any "useful" (killer-) application, although this was not the goal of this thesis.
- Due to computational limitations and due to the differences between my approach and traditional concepts (such as ANN, GAs, etc.), I was unable to provide comparative results.

In summary: this thesis presented basically an explorative study of novel computational and organizational paradigms. I have the feeling that yet another thesis (at least) would be necessary to further develop all approaches, to make extensive comparative tests, and to bring the theories to complete maturity. The path taken basically seems to me a promising path which paves the way to novel and unconventional machines with unique and exciting properties.

8.2 Future Work

This section will feature some general (and rather philosophical) considerations about possible future paths, visions, and ideas.

8.2.1 Artificial Chemistries

Artificial chemistries are an extremely powerful tool to compute in uncertain, irregular, and massively parallel environments. This can maybe best seen by means of the chemical gradients, which spread throughout a network even though there are many broken interconnections. A particle can easily follow a gradient and lost particles can be compensated by sending them in multiple instances. Not all particles will take the same path, so the chance that one arrives at the destination fairly high.

Despite their powerfulness, there is one huge problem with artificial chemistries: no generally applicable method does exist which tells us how to "program" and build artificial chemistries. The his clearly a nontrivial problem which should be address in the near future.

Artificial chemistries are naturally also very interesting to address the challenge of self-replication. Self-replicating molecules are fairly easy to implement as well as artificial genomes which are composed of two strands. Not unlike to the natural *base pairing* (or *nucleotide pairing*), the molecule strands can also be copied by forming complementary strands. In the DNA, A (adenine) always pairs with T (thymine), and C (cytosine) always with G (guanine). The rules of base pairing tell us that if we can read the sequence of nucleotides on one strand of DNA, we can immediately deduce the

complementary sequence on the other strand. They further tell us that in the base composition of DNA the quantity of adenine equals the quantity of thymine and the quantity of guanine equals the quantity of cytosine (known as Chargaff's rule).

Douglas Hofstadter's *Typogenetics* artificial genetic system [194] uses the DNA base pairing mechanisms to copy entire strands. Let us for example assume the pairings as shown in Table 8.1 (the pairing-choice does not really matter). They present a simple shift of 13 steps in the ordered series of symbols in the alphabet.

$$
\begin{array}{l}
\mathsf{A} \longleftrightarrow \mathsf{N} \\
\mathsf{B} \longleftrightarrow \mathsf{O} \\
\mathsf{C} \longleftrightarrow \mathsf{P} \\
\mathsf{D} \longleftrightarrow \mathsf{Q} \\
\mathsf{E} \longleftrightarrow \mathsf{R} \\
\mathsf{F} \longleftrightarrow \mathsf{S} \\
\mathsf{G} \longleftrightarrow \mathsf{T} \\
\mathsf{H} \longleftrightarrow \mathsf{U} \\
\mathsf{I} \longleftrightarrow \mathsf{V} \\
\mathsf{J} \longleftrightarrow \mathsf{W} \\
\mathsf{K} \longleftrightarrow \mathsf{X} \\
\mathsf{L} \longleftrightarrow \mathsf{Y} \\
\mathsf{M} \longleftrightarrow \mathsf{Z}
\end{array}
$$

Table 8.1: An example of symbol pairings.

Using these pairings, the strand S is transformed into the strand $\bar{S}$:

$$
\begin{aligned}
S &= (-\mathsf{BEG}-\mathsf{HUS}-\mathsf{USO}-\mathsf{BHS}-\mathsf{HUS}-\mathsf{END}-) \\
\bar{S} &= (-\mathsf{ORT}-\mathsf{UHF}-\mathsf{HFB}-\mathsf{OUF}-\mathsf{UHF}-\mathsf{RAQ}-) \\
\bar{\bar{S}} = S &= (-\mathsf{BEG}-\mathsf{HUS}-\mathsf{USO}-\mathsf{BHS}-\mathsf{HUS}-\mathsf{END}-)
\end{aligned}
$$

Both writings can be used and their meaning is identical. Figure 8.1 schematically illustrates how a molecular strand can be copied via its complementary strand.

Such possibilities open of course highly interesting paths. The main problem of such an approach is the impossibility to do anything useful on todays computers. Physical or chemical simulations which include that much details are most often computationally intractable.

However, artificial chemistries must not necessary be limited to computer science. One might also imagine artificially engineered but real chemistries, which might maybe someday help to realize one of Stuart

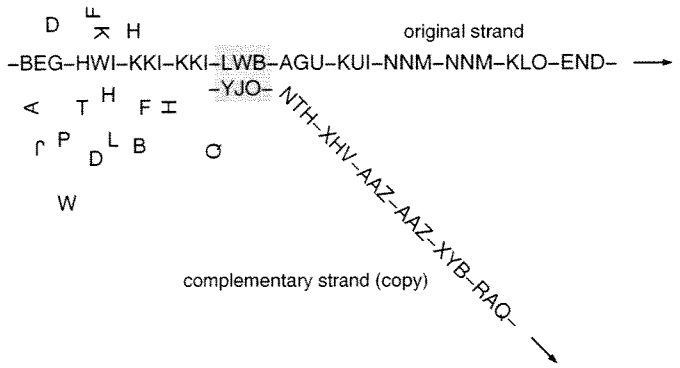

Figure 8.1: Copying a molecular strand.

Kauffman's dreams[1]:

> "An autonomous agent is something that can both reproduce itself and do at least one thermodynamic work cycle. It turns out that this is true of all free-living cells, excepting weird special cases. They all do work cycles, just like the bacterium spinning its flagellum as it swims up the glucose gradient. The cells in your body are busy doing work cycles all the time.
>
> Once I had this definition, my next step was to create and write about a hypothetical chemical autonomous agent. It turns out to be an open thermodynamic chemical system that is able to reproduce itself and, in doing so, performs a thermodynamic work cycle. I had to learn about work cycles, but it's just a new class of chemical reaction networks that nobody's ever looked at before. People have made self-reproducing molecular systems and molecular motors, but nobody's ever put the two together into a single system that is capable of both reproduction and doing a work cycle."

8.2.2 Random and Imperfect Structures

In a yet unpublished paper [265], Mesot and I were able to mathematically proof and to verify experimentally that random boolean networks are able to solve the density and firefly synchronization task much better than regular cellular automata. The main reason for this is the following insight: both the density as well as the firefly synchronization task are "global" problems, which must be solved by local interactions only. Each cell within a of a cellular automata, however, has a highly local and biased view on the entire CA as all cells are regularly interconnected to the local neighborhood.

[1] From a recent talk on `edge.org`

Random boolean network, on the other hand, do have also have a local but an *unbiased* view on the entire set of cells as connections are randomly established. For the density task, for example, the number of 1's in all cells is a crucial information. In CA's the global statistical distribution of the number of ones is biased when viewed from a single cell. This is not the case for random boolean networks where the distribution remains the same. The unbiased view on the global system state is a crucial element for quickly finding a successful solution to this kind of problem.

I mentioned this example here to show that random structures can be more powerful for certain problems than their regular counterparts. Whereas years of research around the density and firefly synchronization task based on cellular automata have revealed fairly little insight and provided no general theory at all, [265] provides a mathematical approach to solve the problem with random boolean networks. This also means that one can make valid predictions and statements, for example regarding the "quality" of a system or regarding its convergence towards the perfect solution.

Research in the domain of networks in general has become fairly popular recently (see for example [29,53] for introductory reading), last but not least because of the internet. However, a lot remains to be done.

One scenario seems fairly clear for the future (as stated in the introduction already):

> "Molecular electronics can be developed if we are able to program a random arrangement of molecules or a field-programmable molecular random array" [366].

However, as we have seen in the previous chapter, the basic elements of the amorphous computer, i.e., the amorphons, are fairly complex. It would therefore — at least in my humble opinion — be totally wrong say that what has been proposed in this thesis might be used in future nanostructures. These two worlds are very different and an amorphon cannot be compared with a nano-transistor (occupying a few atoms only, maybe) which will be located on a huge grid of similar components.

Today, a liquid crystal display commonly offers resolutions of, say about 1280×854 dots. This means that there are $1,093,120$ elements (= 1 dot = three transistors in general) which almost perfectly work (at the exception of handful of dots at most). It would be impossible to perfectly assemble and operate such huge number of elements without sophisticated fault-tolerant mechanisms. I mention this here to make clear that the technology for creating (almost) perfectly functioning systems on the basis of imperfect substrates is available already. However, the approach is most probably not bio-inspired at all (but at least it works) and it is very unlikely that the approach developed for regular structures can be straightforwardly applied

to random structures. The two domains are too different and each one requires new and different methods.

8.2.3 The Outer Reaches of Computer Science...

The design of intelligent machines is a subject that has already interested philosophers and researchers for a very long time and remains a hot topic. The first 50 years of general computation and AI, which roughly spanned the second half of the twentieth century, were characterized by extravagant swings between giddy overstatement and embarrassing near-paralysis. Today, many a philosopher and humanist thinker is convinced that the quest for *artificial intelligence* (*AI*) or *machine intelligence* has turned out to be a failure since computational approaches do not have the performance, flexibility and reliability of neural information processing systems [459]. At least, no artificial system created so far is able to come close to human-like intelligence, and even the most complex artificial systems created seem much simpler — at least at first glance — than their natural counterparts, although this is much debated. For example, is a large airplane more or less complex than a natural cell? Mankind has certainly constructed extremely impressive computing systems and extremely large and complex networks (e.g., the Internet) that perform operations that no human brain could ever complete. But are these systems really intelligent? All "intelligent" systems constructed so far are in the domain of algorithms and thus in the computational domain of Turing machines. Nobody so far has built a physical machine that is able to perform computations a Turing machine could not perform (despite some hypercomputationalists' claims).

Today, machines clearly lack common sense and there seems to be no research consensus on how to give it to them. People have started thinking about the possibility that simulating the mind in silicon might be impossible — or at least impossible using today's methods. Should we first forget about computers and look closer at the gray stuff in the brain, since the actual knowledge of the brain is severely fragmented and many details are still not at all understood?

Alan Kay, a visionary computer scientist and one of the founders of the Xerox Palo Alto Research Center, probably hit the spot with the following statement: "The computer 'revolution' hasn't happened yet!"[2] Kay also postulated that the ideal computer would function like a living organism. Each cell would behave in accord with others to accomplish an end goal, but would also be able to function autonomously. So far so good, but will computer science be enough? Using the term of philosopher Thomas Kuhn,

[2] Talk given at Educom 1998,
`http://www.educause.edu/conference/e98/webcast98.html`.

we might just need a "paradigm shift" [227], something different to anything anyone has already thought of.

I made a very modest attempt in that direction with this thesis. A lot remains to be done! My personal conviction, however, is fairly well mirrored by a recent statement by Freeman Dyson:

> "Anything simple enough to be understandable will not be complicated enough to behave intelligently, while anything complicated enough to behave intelligently will not be simple enough to understand."
>
> — Freeman Dyson
> "What's your law?, *The World Question Center*,
> `www.edge.org`, 2004

MATLAB Amorphous Membrane Blending Toolbox Function Reference

The following list provides an overview on the MATLAB Amorphous Membrane Blending toolbox functions. The list is sorted in alphabetical order. The entries correspond to the output of what is obtained by the MATLAB command `help function`.

```
function cell = addInputToCell(cell, inputs, varargin)
%ADDINPUTTOCELL  Add elements to the cell's inset

function cells = addInputToCellPop(cells, inputs, varargin)
%ADDINPUTTOCELLPOP  Add elements to the cell population

function cell = addMolecules(cell, molecules, varargin)
%ADDMOLECULES Add molecules to the cell

function cell = addReactions(cell, reactions, varargin)
%ADDMOLECULES Add reactions to the cell

function [anb, amorphon] = amorphonEnsCenter(amorphons, posmax,...
  varargin)
%AMORPHONENSCENTER  Center of the amorphon ensemble

function [cells, fmax, fmin, favg, bestID] = ...
  biasCellPopFitness(cells, div, varargin)
%BIASCELLPOPFITNESS  Bias cell population fitness

function cells = blendCellPopulation(cells, varargin)
%BLENDCELLPOPULATION  Blends cell parings within the population

function cell = blendCells(cell1, cell2, varargin)
```

```
%BLENDCELLS  Blends a new cell from two existing cells

function cells = breedCellPopulation(cells, varargin)
%BREEDCELLPOPULATION  Blend cells and eliminate cells

function amorphons = cellCreationEx1(N, membranes, varargin)
%CELLCREATIONEX1  Cell creation experiment 1

function amorphons = cellCreationEx2(membranes, varargin)
%CELLCREATIONEX1  Cell creation experiment 2

function cells = cellPopulationEx1(N, epochs, evolsteps, varargin)
%CELLPOPULATIONEX1  Experiment 1

function beststr = commonStringElements(string1, string2, varargin)
%COMMONSTRINGELEMENTS  Finds common structure elements by elements

function match = comparePolynomes(poly1, poly2, xvalues, varargin)
%PLOTPOLYNOME  Compares the y-values of two polynomes.

function outstring = completeString(string, varargin)
%COMPLETESTRING  Completes the input string with additional symbols

function string = composeString(string1, string2, varargin)
%COMPOSESTRING  Composes the two input strings to a new

function [nodes, edges] = createCAC(N, posmax, kavg, cavg, varargin)
%createCAC  Creates a N-particle circuit amorphous computer topology

function cells = createCellEnsemble(cells, connections, varargin)
%CREATECELLENSEMBLE  Creates a structure-array containing N cells

function amorphons = createMembranes(amorphons, membranes,...
  upcellid, varargin)
%CREATEMEMBRANES  Creates a membrane structure an the amorphons

function graph = displayCAC(coordmatrix, varargin)
% DISPLAYCAC  Displays particles of a circuit amorphous computer

function graph = displayCACGraph(coordmatrix, edges, varargin)
% DISPLAYCACGRAPH  Displays the graph of a circuit amorphous computer

function cells = emptyInOutSets(cells, varargin)
%EMPTYINOUTSETS  Resets in- and outsets of N cells

function [cells, fmax, fmin, favg, bestID] = ...
  evaluateCellPopulation(cells, outset, varargin)
%EVALUATECELLPOPULATION  Evaluates structure-array containing N cells
```

```
function [coord, color] = extractDisplayInformation(amorphons,...
   varargin)
%EXTRACTDISPLAYINFORMATION  Extracts coords and color information

function objects = exractObjects(partofrule)
%EXTRACTOBJECTS  Extracts the objects of a multiset

function common = findStrings(string1, string2, varargin)
%FINDSTRINGS  Find consecutive occurencies of the shorter within

function [meanc, maxc, minc] = getACConnectivity(aswarm)
%GETACCONNECTIVITY  Returns the connectivity of the amorphous
%computer

function [meanc, maxc, minc] = getACData(aswarm)
%GETACDATA  Returns data on the AC

function attr = getCellAttractivity(cell, varargin)
%GETCELLATTRACTIVITY  Cellular attractivity measure

function cellids = getDownCellIDs(membranes, varargin)
%GETDOWNCELLIDS  Return membrane ids of next level

function nbreact = getNbMoleculesEns(aswarm)
%GETNBMOLECULESENS  Returns the number of reactions

function nbreact = getNbReactionsEns(aswarm)
%GETNBREACTIONS  Returns the number of reactions

function[result]=getidstr(extension)
%GETIDSTR  Returns an identification string with a file extension

function[result]=getidstrwe
%GETIDSTR  Returns an identification string without a file extension

function cells = improveCellPopMapping(cells, varargin)
%IMPROVECELLPOPMAPPING  Improves the mappings between the cells

function cell = improvePairings(incell, varargin)
%IMPROVEPAIRINGS  Improves the rule pairings of a cell

function amorphon = initAmorphon(maxrand, varargin)
%INITAMORPHON  Generates structure-array containing amorphon
%information

function amorphons = initAmorphonEnsemble(N, coord, edges, varargin)
%INITAMORPHONENSEMBLE  Generates structure-array containing N
%amorphons
```

```
function agradient = initAmorphonGradient(name, value, varargin)
%INITAMORPHONGRADIENT  Amorphon gradient

function amessage = initAmorphonMessage(name, parameters, message,...
  varargin)
%INITAMORPHONMESSAGE  Amorphon message

function cell = initCell(molecules, reactions, varargin)
%INITCELL Generates structure-array containing cell information

function cells = initCellEnsemble(N, posmax, varargin)
%INITCELLENSEMBLE  Generates structure-array containing N cells

function cells = initCellPopulation(N, varargin)
%INITCELLPOPULATION  Generates a population of N cells

function ioint = initIOInterface(inelements, varargin)
%INITIOINTERFACE  Initialize input/output interface

function cells = initRGCEnsemble(N, posmax, varargin)
%INITRBCENSEMBLE  Generates a random gate chemistry of N cells

function aswarm = initRandAmChemistry(alphabet, specmarkes, nbreact,
  nbmol, mollength, tokenlen, targetlen, nbtokensep, nbtargetsep,...
  posmax, N, vargargin)
%INITRANDAMCHEMISTRY Generates random amorphon chemistry

function cell = initRandomCell(alphabet, specmarkes, nbreact, nbmol,
  mollength, tokenlen, targetlen, nbtokensep, nbtargetsep, posmax,...
  vargargin)
%INITRANDOMCELL  Generates random cell chemistry

function cell = initialPairings(cell1, cell2, varargin)
%INITIALPAIRINGS  Generates a cell that contains rule pairings

function cells = mapCellPopulation(cells, varargin)
%MAPCELLPOPULATION  Maps structure-array containing N cells

function cell = meldPairings(incell, varargin)
%MELDPAIRINGS  Melds the rule pairings of a cell

function rule = meldRules(rule1, rule2, varargin)
%MELDRULES  Melds two rules into a new rule

function cell = plotChemicals(cell, chemicals, steps, varargin)
%PLOTNBCHMICALS  Plots the evolution of the number of chemicals

function void = plotGradient(amorphons, gname, posmax, varargin)
```

```
%PLOTGRADIENT  Plots the gradient on a 3D surface

function reactions = randomBinCompReactions(specmarkers, targetlen,...
  varargin)
%RANDOMBINCOMPREACTIONS  Creates a complete set of random binary
%reactions

function molecules = randomMolecules(alphabet, nbmolecules,...
 mollength, varargin)
%RANDOMMOLECULES Creates a set of random molecules

function reactions = randomReactions(alphabet, specmarkes,...
  nbreactions,tokenlen, targetlen,  nbtokensep, nbtargetsep,...
  varargin)
%RANDOMREACTIONS  Creates a set of random reactions

function cells = resetCellPopulation(cells, varargin)
%RESETCELLPOPULATION  Resets a population of N cells

function similarity = ruleSimilarity(rule1, rule2, varargin)
%RULESIMILARITY  Computes the similarity of two rules

function amorphons = runAmorphonEnsemble(amorphons, varargin)
%RUNAMORPHONENSEMBLE  Decodes and executes the instructions message
%queue

function cell = runCell(cell, varargin)
%RUNCELL  Applies the reactions to the molecules.

function cells = runCellPopulation(cells, varargin)
%RUNCELLPOPULATION  Runs structure-array containing N cells

function cells = searchCellNeighbors3D(cells,searchradius)
%SEARCHCELLNEIGHBORS3D Updates the cell's neighborhood

function [pos, gradient] = searchGradients(amorphon, gname,...
  varargin)
%SEARCHGRADIENT  Returns the gradient value

function similarity = specSymbolSimilarity(string1, string2,...
  varargin)
%SPECSYMBOLSIMILARITY  Computes the similarity of two special
%symbols

function similarity = stringSimilarity(string1, string2, varargin)
%STRINGSIMILARITY  Computes the similarity of two strings

function amorphons = testSearchCellCore(steps, varargin)
%TESTSEARCHCELLCORE  Search cell core test
```

```
function amorphons = updateAmorphonGradients(amorphons, varargin)
%UPDATEAMORPHONGRADIENTS  Spreads the amorphon gradients

function cells = updateCellInset(cells, c)
%UPDATECELLINSET  Search for elemenst in the neighbors outsets
```

List of Figures

List of Tables

List of Algorithms

List of Examples, Theorems, Definitions, Propositions, and Corollaries

Bibliography

[1] International technology roadmap for semiconductors. Semiconductor Industry Association, `http://public.itrs.net/Files/2001ITRS`, 2001.

[2] H. Abelson, D. Allen, D. Coore, C. Hanson, E. Rauch, G. J. Sussman, and R. Weiss. Amorphous computing. *Communications of the ACM*, 43(5):74–82, May 2000.

[3] J. A. Acebron and R. Spigler. Adaptive frequency model for phase-frequency synchronization in large populations of globally coupled nonlinear oscillators. *Physical Review Letters*, 81:2229–2232, 1998.

[4] D. H. Ackley and M. L. Littman. Interactions between learning and evolution. In Langton et al. [236], pages 487–509.

[5] Christoph Adami. *Introduction to Artificial Life*. Springer-Verlag, New York, 1998.

[6] P. Agarwal. *Cell-Based Computer Models in Developmental Biology*. PhD thesis, New York University, Department of Computer Science, September 1993.

[7] I. Aleksander. Random logic nets: Stability and adaptation. *International Journal of Man-Machine Studies*, 5:115–131, 1973.

[8] I. Aleksander. From Wisard to Magnus: A family of weightless virtual neural machines. In J. Austin, editor, *RAM-Based Neural Networks*, volume 9 of *Progress in Neural Processing*. World Scientific, February 1998.

[9] I. Aleksander, W. V. Thomas, and P. A. Bowden. WISARD: A radical step foward in image recognition. *Sensor Review*, 4:120–124, July 1984.

[10] J. T. Allanson. Some properties of randomly connected neural nets. In C. Cherry, editor, *Proceedings of the 3rd London Symposium on Information Theory*, pages 303–313, Butterworths, London, 1956.

[11] S. I. Amari. Characteristics of randomly connected threshold-element networks and network systems. *Proceedings of the IEEE*, 59(1):35–47, January 1971.

[12] S. I. Amari. Learning patterns and pattern sequences by self-organizing nets of threshold elements. *IEEE Transactions on Computers*, C-21(11):1197–1206, November 1972.

[13] M. A. Arbib, editor. *The Handbook of Brain Theory and Neural Networks*. MIT Press, Cambridge, MA, 1995.

[14] W. R. Ashby. *Design for a Brain*. Wiley, New York, 1956.

[15] W. R. Ashby. *An Introduction to Cybernetics*. Chapman and Hall, London, 1956.

[16] W. R. Ashby, H. von Forster, and C. C. Walker. Instability of pulse activity in a net with threshold. *Nature*, 196:561, 1966.

[17] P. J. Ashenden. *The Designer's Guide to VHDL*. Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1996.

[18] J. C. Astor. A developmental model for the evolution of artificial neural networks: Design, implementation, evaluation. Master's thesis, University of Heidelberg, September 3 1998.

[19] J. C. Astor and C. Adami. A developmental model for the evolution of artificial neural networks. *Artificial Life*, 6(3):189–218, 2001.

[20] N. Baas. Emergence, hierarchies, and hyperstructures. In C. G. Langton, editor, *Artificial Life III*, volume XVII of *Santa Fe Institute Studies in the Sciences of Complexity*, pages 515–537, Reading, MA, 1994. Addison-Wesley.

[21] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, New York, 1996.

[22] A. Baddeley. Memory. In *The MIT Encyclopedia of the Cognitive Sciences*, pages 514–517. MIT Press, Cambridge, MA, 1999.

[23] S. Bahar and F. Moss. The nonlinear dynamics of the crayfish mechanoreceptor system. *International Journal of Bifurcation and Chaos*, 13(8):2013–2034, 2003.

[24] P. Bak and L. Chen. Self-organized criticality. *Scientific American*, 265:26–33, January 1991.

[25] P. Ball. Life's lessons in design. *Nature*, 409:413–416, January 18 2001.

[26] J. P. Banâtre, A. Coutant, and D. Le Metayer. A parallel machine for multiset transformation and its programming style. *Future Generation of Computer Systems*, 4:133–144, 1988.

[27] W. Banzhaf, T. Christaller, P. Dittrich, J. T. Kim, and J. Ziegler, editors. *Advances in Artificial Life. Proceedings of the 7^{th} European Conference, ECAL2003*, volume 2801 of *Lecture Notes in Artificial Intelligence*, Berlin, Heidelberg, 2003. Springer-Verlag.

[28] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming–An Introduction: On the Automatic Evolution of Computer Programs and its Applications.* Morgan Kaufmann Publishers, San Francisco, CA, 1997.

[29] A.-L. Barabási. *Linked: The New Science of Networks.* Perseus, Cambridge, MA, 2002.

[30] H. G. Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam, NL, 1984.

[31] W. Bechtel. Decomposing the mind-brain: A long-term pursuit. *Brain and Mind*, 3(2):229–242, 2002.

[32] W. Bechtel and A. Abrahamsen. *Connectionism and the Mind. Parallel Processing, Dynamics, and Evolution of Networks.* Blackwell Publishers Inc., Malden, MA, 1 edition, 1991.

[33] M. Bedau, J. McCaskill, N. Packard, and S. Rasmussen, editors. *Artificial Life VII: Proceedings of the Seventh International Conference on Artificial Life*, Complex Adaptive Systems, Cambridge, MA, 2000. MIT Press.

[34] M. A. Bedau, J. S. McCaskill, N. H. Packard, S. Rasmussen, C. Adami, D. G. Green, T. Ikegami, K. Kaneko, and T. S. Ray. Open problems in artificial life. *Artificial Life*, 6(4):363–376, 2000.

[35] R. D. Beer and J. C. Gallagher. Evolving dynamical neural networks for adaptive behavior. *Adaptive Behavior*, 1(1), 1992.

[36] A. Bell. *Formal Computational Models of Biological Systems.* PhD thesis, University of Sheffield, 1999.

[37] J.-P. Benâtre and D. Le Métayer. The gamma model and its discipline of programming. *Science Comput. Programming*, 15:55–70, 1990.

[38] Y. Benenson, T. Paz-Elizur, R. Adar, E. Keinan, Z. Leivneh, and E. Shapiro. Programmable and autonomous computing machine made of biomolecules. *Nature*, 414(6862):430–434, 2001.

[39] E. R. Berlekamp, J. H. Conway, and R. K. Guy. *Winning Ways for your Mathematical Plays*, volume 2: Games in Particular. Academic Press, London, 1982.

[40] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.

[41] H. Bersini and V. Detours. Asynchrony induces stability in cellular automata based models. In R. A. Brooks and P. Maes, editors, *Proceedings of the Artificial Life IV conference*, pages 382–387, Cambridge, MA, 1994. MIT Press.

[42] M. Bertram and A. S. Mikhailov. Pattern formation on the edge of chaos: Mathematical modeling of CO oxidation on a Pt(100) surface under global delayed feedback. *Physical Review E*, 67:0362071–11, 2003.

[43] H.-G. Beyer. *Theory of Evolution Strategies*. Natural Computing Series. Springer-Verlag, Berlin, Heidelberg, 2001.

[44] E. Bilotta, D. Gross, T. Smith, T. Lenaerts, S. Bullock, H. H. Lund, J. Bird, R. Watson, P. Pantano, L. Pagliarini, H. Abbass, R. Standish, and M. A. Bedau, editors. *Alife VIII-Workshops. Workshop Proceedings of the 8^{th} International Conference on the Simulation and Synthesis of Living Systems*. University of New South Wales, Australia, December 2002.

[45] E. Bilotta, A. Lafusa, and P. Pantano. Searching for complex CA rules with GAs. *Complexity*, 8(3):56–67, 2003.

[46] E. J. W. Boers and H. Kuiper. Biological metaphors and the design of modular artificial neural networks. Master's thesis, Rijksunversität te Leiden, 1992.

[47] H. Bojinov, A. Casal, and T. Hogg. Multiagent control of self-reconfigurable robots. In *Proceedings of the Fourth International Conference on MultiAgent Systems (ICMAS-2000)*, pages 143–150, Los Alamitos, CA, July 10-12 2000. IEEE Computer Society.

[48] S. Bornholdt and T. Rohlf. Topological evolution of dynamical networks: Global criticality from local dynamics. *Physical Review Letters*, 84(26):6114–6117, June 2000.

[49] V. Braitenberg. *Vehicles: Experiments in Synthetic Psychology.* MIT Press, Cambridge, MA, 1984.

[50] J. Breyer, J. Ackermann, and J. S. McCaskill. Evolving reaction-diffusion ecosystems with self-assembling structure in thin films. *Artificial Life*, 4(1):25–40, Winter 1998.

[51] R. Brooks. The relationship between matter and life. *Nature*, 409:409–411, January 18 2001.

[52] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, March 1986.

[53] M. Buchanan. *Small World: Uncovering Nature's Hidden Networks.* Weidenfeld & Nicolson, London, 2002.

[54] J. Buck and E. Buck. Synchronous fireflies. *Scientific American*, 234:74–85, May 1976.

[55] M. Bucolo, R. Caponetto, L. Fortuna, M. Frasca, and A. Rizzi. Does chaos work better than noise? *IEEE Circuits and Systems*, 2(3):4–19, 2002.

[56] J. Busch and W. Banzhaf. How to program artificial chemistries. In Banzhaf et al. [27], pages 20–30.

[57] W. J. Butera. *Programming a Paintable Computer.* PhD thesis, MIT Media Lab, February 2002.

[58] C. Calude and G. Paun. *Computing with Cells and Atoms: An Introduction to Quantum, DNA and Membrane Computing.* Taylor & Francis, New York, 2000.

[59] M. Canella, F. Miglioli, A. Bogliolo, E. Petraglio, and E. Sanchez. Performing DNA comparison on a bio-inspired tissue of FPGAs. In *Proceedings of the* 10^{th} *Reconfigurable Architectures Workshop (RAW03),Nice, France*, April 2003.

[60] M. S. Capcarrere. *Cellular Automata and Other Cellular Systems: Design & Evolution.* PhD Thesis No 2541, Swiss Federal Institute of Technology, Lausanne, 2002.

[61] M. S. Capcarrere. Evolution of asynchronous cellular automata. In J. J. Merelo Guervós, A. Panagiotis, and H.-G. Beyer, editors, *Parallel Problem Solving from Nature*, volume 2439 of *Lecture Notes in Computer Science*, pages 903–912, Berlin, Heidelberg, 2002. Springer-Verlag.

[62] M. S. Capcarrere, A. Tettamanzi, M. Tomassini, and M. Sipper. Statistical study of a class of cellular evolutionary algorithms. *Evolutionary Computation*, 7(3):255–274, 1998.

[63] S. H. Carson, J. B.Peterson, and D. M. Higgins. Decreased latent inhibition is associated with increased creative achievement in high-functioning individuals. *Journal of Personality and Social Psychology*, 85(3):499–506, September 2003.

[64] R. Ceterchi and D. Sburlan. Simulating Boolean circuits with P systems. In A. Alhazov, C. Martín-Vide, and G. Paun, editors, *Proceedings of the MolCoNet Workshop on Membrane Computing (WMC2003)*, volume 28/03, pages 145–160, Tarragona (Spain), 2003. Rovira i Virgili University, Research Group on Mathematical Linguistics.

[65] A. A. Chien. *Concurrent Aggregates (CA): Supporting Modularity in Massively-Parallel Programs.* MIT Press, Cambridge, MA, 1993.

[66] N. Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2:113–124, 1956.

[67] A. Church. A set of postulates for the foundation of logic. *Annals of Mathematics*, 2(33):346–366, 1932.

[68] A. Church. *The Calculi of Lamda-Conversion.* Princeton University Press, Princeton, New Jersey, 1941.

[69] P. S. Churchland and T. J. Sejnowski. *The Computational Brain.* MIT Press, Cambridge, MA, 1992.

[70] G. Ciobanu and D. Paraschiv. Membrane software. A P system simulator. *Fundamenta Informaticae*, 49(1–3):61–66, 2002.

[71] G. Ciobanu and G. Wenyuan. A parallel implementation of the transition P systems. In A. Alhazov, C. Martín-Vide, and G. Paun, editors, *Proceedings of the MolCoNet Workshop on Membrane Computing (WMC2003)*, volume 28/03, pages 169–169, Tarragona (Spain), 2003. Rovira i Virgili University, Research Group on Mathematical Linguistics.

[72] W. J. Clancey. The frame of reference problem in cognitive modeling. In *Proceedings of the Annual Conference of the Cognitive Science Society*, pages 107–114, Hillsdale, NJ, 1989. Erlbaum.

[73] A. Clark. *Microcognition: Philosophy, Cognitive Science, and Parallel Distributed Processing.* A Bradford Book, MIT Press, Cambridge, MA, 1989.

[74] A. Clark and P. Millican, editors. *The Legacy of Alan Turing: Connectionism, Concepts, and Folk Psychology*, volume 2. Oxford University Press Inc., New York, 1996.

[75] W. A. Clark and B. G. Farley. Generalisation of pattern recognition in a self-organising system. In *Proceedings of the Western Joint Computer Conference*, pages 86–91, 1955.

[76] M. Conrad. The brain-machine disanalogy. *BioSystems*, 22:197–213, 1989.

[77] M. Conrad, R. R. Kampfner, K. G. Kirby, E. N. Rizki, G. Schleis, R. Smalz, and R. Trenary. Towards an artificial brain. *BioSystems*, 23:175–218, 1989.

[78] C. Constantinescu. Trends and challenges in VLSI circuit reliability. *IEEE Micro*, 23(4):14–19, July–August 2003.

[79] D. Coore. Establishing a coordinate system on an amorphous computer. Technical Report MIT/LCS/TR-737, MIT Student Workshop on High Performance Computing in Science and Engineering, 1998. Available online: `http://www.swiss.ai.mit.edu/projects/amorphous`.

[80] D. Coore. *Botanical Computing: A Developmental Approach to Generating Interconnect Topologies on an Amorphous Computer*. PhD thesis, MIT Department of Electrical Engineering and Computer Science, February 1999.

[81] D. Coore, R. Nagpal, and R. Weiss. Paradigms for structure in an amorphous computer. Technical Report AI Memo 1614, MIT Artificial Intelligence Laboratory, October 6 1997.

[82] B. J. Copeland. Even Turing machines can compute uncomputable functions. In C. Calude, J. Casti, and M. Dinneen, editors, *Unconventional Models of Computation*, pages 150–164. Springer-Verlag, London, 1998.

[83] B. J. Copeland. Super Turing-machines. *Complexity*, 4(1):30–32, 1998.

[84] B. J. Copeland. Turing's O-machines, Penrose, Searle, and the brain. *Analysis*, 58:128–138, 1998.

[85] B. J. Copeland and D. Proudfoot. On Alan Turing's anticipation of connectionism. *Synthese: An International Journal for Epistemology, Methodology and Philosophy of Science*, 108:361–377, 1996.

[86] B. J. Copeland and D. Proudfoot. Alan Turing's forgotten ideas in computer science. *Scientific American*, 280(4):76–81, April 1999.

[87] B. J. Copeland and R. Sylvan. Beyond the universal Turing machine. *Australasian Journal of Philosophy*, 77(1):46–66, March 1999.

[88] R. Cori, Y. Métivier, and W. Zielonka. Asynchronous mappings and asynchronous cellular automata. *Information and Computation*, 106:159–202, 1993.

[89] T. Dartnall, editor. *Artificial Intelligence and Creativity.* Kluwer Academic Publishers, Dordrecht, 1994.

[90] C. Darwin. *The Origin of Species.* John Murray, London, 1859.

[91] R. Das, J. P. Crutchfield, M. Mitchell, and J. E. Hanson. Evolving globally synchronized cellular automata. In L. J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 336–343, San Francisco, CA, 1995. Morgan Kaufmann.

[92] M. Davis, editor. *The Undecidable.* Raven Press, Hewlett, New York, 1965.

[93] M. Davis. *The Universal Computer.* W. W. Norton, 2000.

[94] M. Davis. The myth of hypercomputation. In Teuscher [409], pages 195–212.

[95] M. de Boer. *Analysis and Computer Generation of Division Patterns in Cell Layers Using Developmental Algorithms.* PhD thesis, University of Utrecht, 1989.

[96] M. de Boer, D. Fracchia, and P. Prusinkiewicz. Analysis and simulation of the development of cellular layers. In Langton et al. [236], pages 465–483.

[97] H. de Garis. *Genetic Programming: GenNets, Artificial Nervous Systems, Artificial Embryos.* PhD thesis, Brussels University, 1992.

[98] H. de Garis, A. Buller, L. de Penning, T. Chodakowski, and D. Decesare. Initial evolution results on CAM-brain machines (CMBs). In G. Dorffner, H. Bischof, and K. Hornik, editors, *Proceedings of the International Conference on Artificial Neural Networks (ICANN'2001)*, volume 2130 of *Lecture Notes in Computer Science*, pages 814–819. Springer-Verlag, Berlin, Heidelberg, 2001.

[99] H. de Garis, A. Buller, M. Korkin, F. Gers, N. Nawa, and M. Hough. ATR's Artificial Brain ("CAM-Brain") Project: A Sample of what Individual "CoDi-1Bit" Model Evolved Neural Net Modules can do with Digital and Analog I/O. In A. Stoica, D. Keymeulen, and J. Lohn, editors, *Proceedings of the First NASA/DOD Workshop on Evolvable*

Hardware, pages 102–110. IEEE Computer Society, Los Alamitos, CA, July, 19-21 1999.

[100] H. de Garis, M. Korkin, F. Gers, and N. E. Nawa M. Hough. Building an Artificial Brain Using an FPGA Based CAM-Brain Machine. *Applied Mathematics and Computation Journal, Special Issue on Artificial Life and Robotics, Artificial Brain, Brain Computing and Brainware*, 111, 2000.

[101] H. de Garis and N. Macias. Initial functional and architectural thoughts for a 2nd generation brain building machine (BM2). Available here: `http://www.cs.usu.edu/~degaris/papers/`, 2001.

[102] F. Dellaert. Toward a biologically defensible model of development. Master's thesis, Case Western Reserve University, Department of Computer Engineering and Science, January 1995.

[103] F. Dellaert and R. D. Beer. Toward an evolvable model of development for autonomous agent synthesis. In R. A. Brooks and P. Maes, editors, *Artificial Life IV, Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, pages 246–257, Cambridge, MA, 1994. A Bradford Book, MIT Press.

[104] F. Dellaert and R. D. Beer. A developmental model for the evolution of complete autonomous agents. In P. Maes, M. Mataric, J.-A. Meyer, J. Pollack, and S. W. Wilson, editors, *From Animals to Animats 4: Proceedings of the 4th International Conference on Simulation of Adaptive Behavior*, pages 393–401, Cambridge, MA, 1996. A Bradford Book, MIT Press.

[105] P. J. Denning and R. M. Metcalfe, editors. *Beyond Calculation. The Next Fifty Years of Computing.* Copernicus, Springer-Verlag, New York, NY, 1997.

[106] B. Derrida and Y. Pomeau. Random networks of automata: A simple annealed approximation. *Europhysics Letters*, 1(2):45–49, 1986.

[107] D. Deutsch. Quantum theory, the Church-Turing principle of the Universal Quantum Computer. *Proceedings of the Royal Society of London*, A400:97–117, 1985.

[108] E. D'Hondt. Exploring the amorphous computing paradigm. Master's thesis, Vrije Universiteit Brussel, Faculteit von de Wetenschappen, Departement Informatica, August 2000.

[109] P. Speroni di Fenizio. Building life without cheating. Generating boundaries with self-maintaining sets. Master's thesis, University of Sussex, Brighton, UK, 1999.

[110] P. Speroni di Fenizio. A less abstract artificial chemistry. In Bedau et al. [33], pages 49–52.

[111] P. Dittrich. *On Artificial Chemistries.* PhD thesis, University of Dortmund, Department of Computer Science, D-44221 Dortmund, Germany, January 25 2001.

[112] P. Dittrich, J. Ziegler, and W. Banzhaf. Artificial chemistries–a review. *Artificial Life*, 7(3):225–275, 2001.

[113] Georg Dorffner, editor. *Neural Networks and a New AI.* Chapman & Hall, London, 1994.

[114] A. Dorin and J. McCormack. Self-assembling dynamical hierarchies. In Bilotta et al. [44], pages 423–428.

[115] D.-Z. Du and K.-I. Ku. *Theory of Computational Complexity.* John Wiley & Sons, Inc., New York, 2000.

[116] Z. Duan, M. Holcombe, and A. Bell. A logic for biological systems. *Biosystems*, 55(1-3):93–105, February 2000.

[117] L. J. K. Durbeck and N. J. Macias. The Cell Matrix: An architecure for nanocomputing. *Nanotechnology*, 12:217–230, 2001.

[118] S. F. Edwards and P. W. Anderson. Theory of spin glasses. *Journal of Physics F*, 5:965, 1975.

[119] P. Eggenberger. Creation of neural networks based on developmental and evolutionary principles. In W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicod, editors, *Proceedings of the International Conference on Artificial Neural Networks ICANN'97, Lausanne, Switzerland*, volume 1327 of *Lecture Notes in Computer Science*, pages 337–342, Berlin, Germany, 1997. Springer-Verlag.

[120] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing.* Springer-Verlag, Berlin, Heidelberg, 2003.

[121] M. Eigen. The quasispecies model. *Naturwissenschaften*, 58:465–523, 1971.

[122] S. Eilenberg. *Automata, Languages, and Machines*, volume A of *Pure and Applied Mathematics.* Academic Press, 1974.

[123] J. L. Elman, E. A. Bates, M. H. Johnson, A. Karmiloff-Smith, D. Parisi, and K. Plunkett. *Rethinking Innateness. A Connectionist Perspective on Development.* A Bradford Book, MIT Press, Cambridge, MA; London, UK, 1996.

[124] D. Endy and R. Brent. Modelling cellular behaviour. *Nature*, 409(6818):391–395, January 18 2001.

[125] G. Ernst and A. Newell. *GPS: A Case Study in Generality and Problem Solving.* Academic Press, New York, 1969.

[126] B. G. Farley and W. A. Clark. Simulation of self-organising systems by digital computer. *Institute of Radio Engineers Transactions on Information Theory*, 4:76–84, 1954.

[127] G. Fauconnier. *Mental Spaces: Aspects of Meaning Construction in Natural Language.* Cambridge University Press, Cambridge, UK, 1994. First published in 1985 by the Massachusetts Institute of Technology.

[128] G. Fauconnier. *Mappings in Thought and Language.* Cambridge University Press, Cambridge, UK, 1997.

[129] G. Fauconnier and M. Turner. Conceptual integration networks. *Cognitive Science*, 22(2):133–187, April–June 1998.

[130] G. Fauconnier and M. Turner. *The Way We Think: Conceptual Blending and the Mind's Hidden Complexities.* Basic Books, 2002.

[131] R. Field and M. Burger. *Oscillations and Traveling Waves in Chemical Systems.* Wiley, New York, 1985.

[132] K. W. Fleischer. *A Multiple-Mechanism Developmental Model for Defining Self-Organizing Geometric Structures.* PhD thesis, California Institute of Technology, Pasadena, CA, 1995.

[133] K. W. Fleischer and A. H. Barr. A simulation testbed for the study of multicellular development: The multiple mechanisms of morphogenesis. In C. G. Langton, editor, *Artificial Life III*, volume XVII of *SFI Studies in the Science of Complexity*, pages 389–416, Redwood City, CA, 1994. Addison-Wesley.

[134] R. W. Floyd and R. Beigel. *The Language of Machines: An Introduction to Computability and Formal Languages.* Computer Science Press, New York, 1994.

[135] J. A. Fodor. *Psychosemantics.* MIT Press, Cambridge, MA, 1987.

[136] D. B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* IEEE Press, Piscataway, NJ, second edition, 1999.

[137] W. Fontana. Algorithmic chemistry. In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Artificial Life II*, Santa Fe Institute Studies in the Sciences of Complexity, pages 159–210, Redwood City, CA, 1992. Addison-Wesley.

[138] W. Fontana and L. W. Buss. 'The arrival of the fittest': Toward a theory of biological organization. *Bulletin of Mathematical Biology*, 56:1–64, 1994.

[139] W. Fontana and L. W. Buss. What would be conserved if 'the tape were played twice'? *Proceedings of the National Academy of Science, USA*, 91:757–761, 1994.

[140] W. Fontana and Leo W. Buss. The barrier of objects: From dynamical systems to bounded organizations. In J. Casti and A. Karlqvist, editors, *Boundaries and Barriers*, pages 56–116. Addison-Wesley, 1996.

[141] W. Fontana, G. Wagner, and L. W. Buss. Beyond digital naturalism. *Artificial Life*, 1/2:211–227, 1994.

[142] K. M. Ford and P. J. Hayes. *Reasoning Agents in a Dynamic World: The Frame Problem*, volume 1 of *Advances in Human and Machine Cognition*. JAI Press, Greenwich, Connecticut, 1991. Originally presented at the First International Workshop on Human & Machine Cognition, Pensacola, Florida, May 11–13, 1989.

[143] K. M. Ford and P. J. Hayes. On computational wings: Rethinking the goals of artificial intelligence. *Scientific American Presents: Exploring Intelligence*, 9(4), Winter 1998.

[144] S. De Franceschi and L. Kouwenhoven. Electronics and the single atom. *Nature*, 417:701–702, June 13 2002.

[145] S. Franklin and A. Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In J. P. Müller, M. J. Wooldridge, and N. R. Jennings, editors, *Intelligent Agents III: Agent Theories, Architectures, and Languages*, volume 1193 of *Lecture Notes in Computer Science*. Springer-Verlag Berlin Heidelberg, 1997.

[146] R. M. French. The Turing Test: The first fifty years. *Trends in Cognitive Sciences*, 4(3):115–121, 2000.

[147] K. Fukushima. Cognitron: A self-organizing multilayered neural network model. *Biological Cybernetics*, 20:121–136, 1965.

[148] K. Fukushima, S. Miyake, and T. Ito. Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:826–834, 1983.

[149] K. Fukushima and N. Wake. Handwitten alphanumeric character recognition by the Neocognitron. *IEEE Transactions on Neural Networks*, 2(3):355–365, 1991.

[150] P. Gács. Self-correcting two-dimensionnal arrays. In Silvio Micali, editor, *Randomness in computation*, volume 5 of *Advances in Computing Research*, pages 223–326, Greenwich, Conn, 1989. JAI Press.

[151] P. Gács. Reliable cellular automata with self-organization. In *Proceedings of the 38th IEEE Symposium on the Foundation of Computer Science*, pages 90–99, 1997.

[152] A. Galton. The Church-Turing thesis: Its nature and status. In P. Millican and A. Clark, editors, *The Legacy of Alan Turing: Machines and Thought*, volume 1, chapter 8, pages 137–164. Oxford University Press Inc., New York, 1996.

[153] T. Gánti. Organization of chemical reactions into dividing and metabolizing units: The chemotons. *BioSystems*, 7:15–21, 1975.

[154] T. Gánti. Biogenesis itself. *Journal of Theoretical Biology*, 187:583–593, 1997.

[155] L. Garber and D. Sims. In pursuit of hardware-software codesign. *IEEE Computer*, 31(6):12–14, June 1998.

[156] H. Gardner. *Intelligence Reframed: Multiple Intelligences for the 21st Century*. Basic Books, New York, NY, 1999.

[157] C. Gershenson. Phase transitions in random boolean networks with different updating schemes. Submitted to Physica D. Available online: `http://arxiv.org/abs/nlin.AO/0311008`.

[158] C. Gershenson. Classification of random boolean networks. In Standish et al. [385], pages 1–8.

[159] M. Giacobini, E. Alba, and Marco Tomassini. Selection intensity in asynchronous cellular evolutionary algorithms. In E. Cantú-Paz, J. A. Foster, K. Deb, L. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. K. Standish, G. Kendall, S. W. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. A. Potter, A. C. Schultz, K. A. Dowsland, N. Jonoska, and J. F. Miller, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003)*, volume 2723 of *Lecture Notes in Computer Science*, pages 955–966, Berlin, Heidelberg, 2003. Springer-Verlag.

[160] W. W. Gibbs. Cybernetic cells. *Scientific American*, 285(2):43–47, August 2001.

[161] L. Glass and M. C. Mackey. *From Clocks to Chaos: The Rhythms of Life*. Princeton University Press, 1988.

[162] N. S. Goel and R. L. Thompson. Movable Finite Automata (MFA): A new tool for computer modeling of living systems. In Langton [233], pages 317–340.

[163] J. Goguen. An introduction to algebraic semiotics, with applications to user interface design. In Nehaniv [287], pages 242–291.

[164] S. Grand. *Creation: Life and How to Make It.* Weidenfeld & Nicolson, London, 2000.

[165] R. O. Grondin, W. Porod, C. M. Loeffler, and D. K. Ferry. Synchronous and asynchronous systems of threshold elements. *Biological Cybernetics*, 49:1–7, 1983.

[166] D. Gross and McMullin B. The creation of novelty in artificial chemistries. In Standish et al. [385], pages 400–408.

[167] D. Gross and T. Lenaerts. Towards a definition of dynamical hierarchies. In Bilotta et al. [44], pages 45–53.

[168] D. Gross and B. McMullin. Is it the right *Ansatz*? *Artificial Life*, 7(4):355–365, 2001.

[169] F. Gruau. Cellular encoding of genetic neural networks. Technical Report 92-21, Ecole Normale Supérieure de Lyon, Institut IMAG, 1992.

[170] F. Gruau. Genetic systems of boolean neural networks with a cell rewriting developmental process. In D. Whitley and S. D. Schaffer, editors, *Combination of Genetic Algorithms and Neural Networks.* IEEE Computer Society Press, Los Alamitos, CA, 1992.

[171] F. Gruau. *Neural Network Synthesis Using Cellular Encoding and the Genetic Algorithm.* PhD thesis, Ecole Normale Supérieure de Lyon, 1994.

[172] F. Gruau, Y. Lhuillier, P. Reitz, and O. Temam. BLOB computing. Technical report, LRI - University Paris South, April 2003.

[173] F. Gruau and P. Malbos. The Blob: A basic topological concept for hardware-free distributed computation. In C. Calude, M. J. Dinneen, and F. Peper, editors, *Unconventional Models of Computation*, volume 2509 of *Lecture Notes in Computer Science*, pages 151–163, Berlin, Heidelberg, 2002. Springer-Verlag.

[174] F. Gruau and D. Whitley. The cellular developmental of neural networks: The interaction of learning and and evolution. Technical Report 93-04, Ecole Normale Supérieure de Lyon, Institut IMAG, Januar 1993.

[175] Y. H. Gunther, editor. *Essays on Nonconceptual Content.* A Bradford Book, MIT Press, Cambridge, MA, 2003.

[176] P. C. Haddow and P. van Remortel. From here to there: Future robust EHW technologies for large digital designs. In D. Keymeulen, A. Stoica, J. Lohn, and R. Zebulum, editors, *Proceedings of the Third NASA/DoD Workshop on Evolvable Hardware, EH-2001*, pages 232–239. IEEE Computer Society, Los Alamitos, CA, 2001.

[177] H. Haken. *Brain Dynamics, Synchronization and Activity Patterns in Pulse-Coupled Neural Nets with Delays and Noise.* Springer Series in Synergetics. Springer-Verlag, 2002.

[178] P. J. B. Hancock. *Coding Strategies for Genetic Algorithms and Neural Nets.* PhD thesis, Department of Computing Science and Mathematics, University of Stirling, 1992.

[179] M. Hannenbauer. Basics of collaborative problem solving. In *Autonomous Dynamic Reconfiguration*, volume 2427 of *Lecture Notes in Artificial Intelligence*, pages 9–24. Springer-Verlag, Berlin, Heidelberg, Germany, 2002.

[180] S. Harnad. The symbol grounding problem. *Physica D*, 42:353–346, 1990.

[181] H. Hartman and Gérard Y. Vichniac. Inhomogeneous cellular automata (INCA). In E. Bienenstock et al., editor, *Disordered Systems and Biological Organization*, volume F 20, pages 53–57. Springer-Verlag, Berlin, 1986.

[182] I. Harvey and T. Bossomaier. Time out of joint: Attractors in asynchronous random boolean networks. In P. Husbands and I. Harvey, editors, *Proceedings of the Fourth European Conference on Artificial Life*, pages 67–75. MIT Press, Cambridge, MA, 1997.

[183] B. Hasslacher. Beyond the Turing machine. In Herken [188], pages 387–402.

[184] M. H. Hassoun. *Fundamentals of Artificial Neural Networks.* A Bradford Book, MIT Press, Cambridge, MA; London, UK, 1995.

[185] S. Haykin. *Neural Networks: A Comprehensive Foundation.* Prentice Hall, New Jersey, second edition, 1999.

[186] J. R. Heath, P. J. Kuekes, G. S. Snider, and R. S. Williams. A defect-tolerant computer architecture: Opportunities for nanotechnology. *Science*, 280(5370):1716–1721, June 12 1998.

[187] D. Hebb. *The Organization of Behavior.* John Wiley, New York, 1949.

[188] R. Herken, editor. *The Universal Turing Machine: A Half–Century Survey.* Springer-Verlag, Wien, second edition, 1995.

[189] M. Hiratsuka, T. Aoki, and T. Higuchi. Enzyme transistor circuits for reaction-diffusion computing. *IEEE Transactions on Circuits and Systems I*, 46(2):294–303, 1999.

[190] A. Hjelmfelt, E. D. Weinberger, and J. Ross. Chemical implementation of neural networks and turing machines. *Proc. Natl. Acad. Sci. USA*, 88(24):10983–7, December 15 1991.

[191] A. Hjelmfelt, E. D. Weinberger, and J. Ross. Chemical implementation of finite-state machines. *Proc. Natl. Acad. Sci. USA*, 89(1):383–387, January 1 1992.

[192] A. Hodges. Alan Turing and the Turing machine. In Herken [188], pages 3–14.

[193] D. Hofstadter. How could a COPYCAT ever be creative? In Dartnall [89].

[194] D. R. Hofstadter. *Gödel, Escher, Bach: An Eternal Golden Braid.* Basic Books, New York, 1979.

[195] D. R. Hofstadter. *Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought.* Basic Books, New York, 1995.

[196] Jenny Hogan. Quantum bits and silicon chips. *Nature*, 424(6948):484–486, July 31 2003.

[197] J. H. Holland. *Adaption in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, MI, 1975.

[198] J. H. Holland. *Emergence: From Chaos to Order.* Perseus Books, 1999.

[199] F. T. Hong. Molecular electronics: Science and technology for the future. *IEEE Engineering in Medicien and Biology*, 13(1):25–32, February/March 1994.

[200] A. A. Hopgood. Artificial intelligence: Hype or reality? *Computer*, 36(5):24–28, May 2003.

[201] W. Hordijk. The structure of the synchonizing-CA landscape. Technical Report 96-10-078, Santa Fe Institute, Santa Fe, NM (USA), 1996.

[202] J. R. Hurford. Random boolean nets and features of language. *IEEE Transactions on Evolutionary Computation*, 5(2):111–116, April 2001.

[203] Tim J. Hutton. Evolvable self-replicating molecules in an artificial chemistry. *Artificial Life*, 8(4):341–356, 2002.

[204] D. C. Ince, editor. *Collected Works of A. M. Turing: Mechanical Intelligence*. North-Holland, Amsterdam, 1992.

[205] B. Indurkhya. An algebraic approach to modeling creativity of metaphor. In Nehaniv [287], pages 292–306.

[206] T. E. Ingerson and R. L. Buvel. Structure in asynchronous cellular automata. *Physica D*, 10(1–2):59–68, January 1984.

[207] E. Ising. Beitrag zur Theorie des Ferromagnetismus. *Zeitschrift für Physik*, 31:253–258, 1925.

[208] J. S. Judd. Learning in neural networks is hard. In M. Caudill and C. Butler, editors, *First IEEE International Conference on Neural Networks*, volume 2, pages 685–692, San Diego, 1987. IEEE, New York.

[209] J. S. Judd. *Neural Network Design and the Complexity of Learning*. A Pradford Book, MIT Press, Cambridge, MA, 1990.

[210] P. Jung, A. Cornell-Bell, F. Moss, S. Kadar, J. Wang, and K. Showalter. Noise sustained waves in subexcitable media: From chemical waves to brain waves. *Chaos*, 8:567–575, 1998.

[211] Y. Kanada. Asynchronous 1D cellular automata and the effects of fluctuation and randomness. In R. A. Brooks and P. Maes, editors, *ALife IV: Proceedings of the Fourth Conference on Artificial Life*, page Poster, Cambridge, MA, 1994. MIT Press.

[212] S. A. Kauffman. Metabolic stability and epigenesis in randomly connected genetic nets. *Journal of Theoretical Biology*, 22:437–467, 1968.

[213] S. A. Kauffman. *The Origins of Order: Self–Organization and Selection in Evolution*. Oxford University Press, New York; Oxford, 1993.

[214] S. A. Kauffman. *At Home in the Universe*. Oxford University Press, New York; Oxford, 1995.

[215] S. A. Kauffman. *Investigations*. Oxford University Press, New York; Oxford, 2000.

[216] J. Kennedy. Thinking is social: Experiments with the adaptive cultural model. *The Journal of Conflict Resolution*, 42(1):56–76, February 1998.

[217] J. Kennedy and R. C. Eberhart. *Swarm Intelligence.* Morgan Kaufmann Publishers, 2001.

[218] H. Kitano. Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, 4:461–476, 1990.

[219] H. Kitano. Building complex systems using developmental process: An engineering approach. In M. Sipper, D. Mange, and A. Pérez-Uribe, editors, *Proceedings of the Second International Conference on Evolvable Systems (ICES'98)*, volume 1478 of *Lecture Notes in Computer Science*, pages 218–229, Berlin, Germany, September 1998. Springer-Verlag.

[220] S. C. Kleene. Formalized recursive functionals and formalized realizability. In *Memoirs of the American Mathematical Society*, volume 89, 1969.

[221] J. Kodjabachian and J. A. Meyer. Development, learning and evolution in animats. In P. Gaussier and J.-D. Nicoud, editors, *Proceedings of PerAc'94: From Perception to Action*, Los Alamitos, CA, 1994. IEEE Computer Society Press.

[222] M. Korkin, H. de Garis, F. Gers, and H. Hemmi. CBM (CAM-Brain Machine): A Hardware Tool which Evolves a Neural Net Module in a Fraction of a Second and Runs a Million Neuron Artificial Brain in Real Time. In *Proceedings of the Genetic Programming Conference, GP'97*, Stanford, USA, July 1997.

[223] M. Korkin, N. E. Nawa, and D. de Garis. A "spike interval information coding" representation of ATR's CAM-brain machine (CBM). In M. Sipper, D. Mange, and A. Pérez-Uribe, editors, *Proceedings of the Second International Conference on Evolvable Systems (ICES'98)*, volume 1478 of *Lecture Notes in Computer Science*, pages 256–267, Lausanne, Switzerland, September 1998. Springer-Verlag, Berlin, Germany.

[224] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, 1992.

[225] J. R. Koza, F. H. Bennett, D. Andre, and M. A. Keane. *Genetic Programming III: Darwinian Invention and Problem Solving.* Morgan Kaufmann, San Francisco, CA, 1999.

[226] A. Kubík. Toward a formalization of emergence. *Artificial Life*, 9(1):41–65, 2003.

[227] T. S. Kuhn. *The Structure of Scientific Revolutions.* University of Chicago Press, Chicago, 1962.

[228] D. Kuske. Emptiness is decidable for asynchronous cellular machines. In C. Palamidessi, editor, *CONCUR 2000*, Lecture Notes in Computer Science, LNCS 1877, pages 536–551, Berlin, 2000.

[229] J. Lach, W. Mangione-Smith, and M. Potkonjak. Low overhead fault-tolerant FPGA systems. *IEEE Transactions on VLSI Systems*, 6(2):212–221, 1998.

[230] G. Lakeoff and R. E. Núñez. *Where mathematics comes from : how the embodied mind brings mathematics into being.* Basic Books, New York, 2000.

[231] P. K. Lala. *Self-Checking and Fault-Tolerant Digital Design.* Morgan Kaufmann Publishers, San Francisco, 2001.

[232] C. G. Langton. Self-reproduction in cellular automata. *Physica D*, 10:135–144, 1984.

[233] C. G. Langton, editor. *Artificial Life. Proceedings of the Artificial Life Workshop held September, 1987 in Los Alamos, New Mexico*, volume VI of *Santa Fe Institute Studies in the Sciences of Complexity.* Addison-Wesley, Redwood City, CA, 1989.

[234] C. G. Langton, editor. *Artificial Life. Proceedings of the Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems*, Redwood City, CA, 1989. Addison-Wesley.

[235] C. G. Langton, editor. *Artificial Life: An Overview.* MIT Press, Boston, 1997.

[236] C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors. *Artificial Life II.* Addison-Wesley, Redwood City, CA, 1992.

[237] S. Laurence and E. Margolis. Concepts and cognitive science. In Margolis and Laurence [249], chapter 1, pages 3–81.

[238] T. Lenaerts, D. Gross, and R. Watson. On the modelling of dynamical hierarchies. In Bilotta et al. [44], pages 37–44.

[239] A. Lindenmayer. Mathematical models for cellular interaction in development, parts I and II. *Journal of Theoretical Biology*, 18:280–315, 1968.

[240] Q. Luo, C. Perry, D. Peng, Z. Jin, D. Xu, G. Ding, and S. Xu. The neural substrate of analogical reasoning: an fMRI study. *Cognitive Brain Research*, 17:527–534, 2003.

[241] N. J. Macias. The PIG paradigm: the design and use of a massively parallel fine grained self-reconfigurable infinitely scalable architecture. In A. Stoica, D. Keymeulen, and J. Lohn, editors, *Proceedings of the First NASA/DOD Workshop on Evolvable Hardware*, pages 175–180. IEEE Computer Society, Los Alamitos, CA, 1999.

[242] N. J. Macias and L. J. K. Durbeck. Self-assembling circuits with autonomous fault handling. In A. Stoica, J. Lohn, R. Katz, D. Keymeulen, and R. S. Zebulum, editors, *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware*, pages 46–55, Los Alamitos, CA, 2002. IEEE Computer Society.

[243] M. Madhu, V. S. Murty, and K. Krithivasan. A hardware realization of P systems with carriers. Poster presentation at the Eight International Conference on DNA based Computers, Hokkaido University, Sapporo Campus, Japan, June 10–13 2002.

[244] N. R. Mahapatra and S. Dutt. Hardware-efficient and highly reconfigurable 4- and 2-track fault-tolerant designs for mesh-connected arrays. *Journal of Parallel and Distributed Computing*, 61(10):1391–1411, October 2001.

[245] D. Mange, M. Sipper, and P. Marchal. Embryonic electronics. *Biosystems*, 51(3):145–152, September 1999.

[246] D. Mange, M. Sipper, A. Stauffer, and G. Tempesti. Toward robust integrated circuits: The embryonics approach. *Proceedings of the IEEE*, 88(4):516–540, April 2000.

[247] D. Mange, A. Stauffer, G. Tempesti, and C. Teuscher. Dispositif électronique à affichage électro-optique commandé par des circuits logiques programmables. Demande de brevet européen No 01201221.7, March 29, 2001.

[248] D. Mange and M. Tomassini, editors. *Bio-Inspired Computing Machines: Towards Novel Computational Architectures.* Presses Polytechniques et Universitaires Romandes, Lausanne, Switzerland, 1998.

[249] E. Margolis and S. Laurence, editors. *Concepts: Core Readings.* A Bradford Book, MIT Press, Cambridge, MA, 1999.

[250] C. Martín-Vide, G. Paun, J. Pazos, and A. Rodríguez-Patón. Tissue P systems. *Theoretical Computer Science*, 296(2):295–326, March 8 2003.

[251] D. Martland. Auto-associative pattern storage using synchronous boolean networks. In *Proceedings of the First IEEE International Con-*

ference on Neural Networks, volume III, pages 355–366, San Diego, CA, 1987.

[252] D. Martland. Behaviour of autonomous, (synchronous) boolean networks. In *Proceedings of the First IEEE International Conference on Neural Networks*, volume II, pages 243–250, San Diego, CA, 1987.

[253] D. Martland. Configurable boolean networks. In L. Personnaz and G. Dreyfus, editors, *Neural Networks from Models to Applications. Proceedings of the First European Conference on Neural Networks, nEuro'88*, volume III, pages 355–366, IDSET, Paris, June 1988.

[254] D. Martland. Dynamic behavior of boolean networks. In I. Aleksander, editor, *Neural Computing Architectures: The Design of Brain-Like Machines*, chapter 11, pages 217–235. North Oxford Academic, London, 1989.

[255] M. Mataric. Issues and approaches in the design of collective autonomous agents. *Robotics and Autonomous Systems*, 16(2–4):321–331, 1995. December.

[256] N. Mathur. Beyond the silicon roadmap. *Nature*, 419(6907):573–575, October 10 2002.

[257] B. Mayer and S. Rasmussen. Self-reproduction of dynamical hierarchies in chemical systems. In C. Adami, R. K. Belew, H. Kitano, and C. E. Taylor, editors, *Proceedings of the* 6^{th} *International Conference on Artificial Life*, pages 123–129, Cambridge, MA, 1998. A Bradford Book, MIT Press.

[258] E. Mayoraz. *Feedforward Boolean Neural Networks with Discrete Weights: Computational Power and Training.* PhD thesis, Swiss Federal Institute of Technology Lausanne (EPFL), CH–1015 Lausanne, 1993. Thesis No 1157.

[259] W. McCarthy. *Hacking Matter: Levitating Chairs, Quantum Mirages and the Infinite Weirdness of Programmable Atoms.* Basic Books, New York, 2003.

[260] M. K. McClintock. Menstrual synchrony and suppression. *Nature*, 229:244–245, 1971.

[261] W. S. McCulloch and W. H. Pitts. A logical calculus of the ideas immanent in neural nets. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[262] B. McMullin and F. Varela. Rediscovering computational autopoiesis. In P. Husbands and I. Harvey, editors, *Proceedings of the Fourth European Conference on Artificial Life*, pages 38–47, Cambridge, MA, 1997. MIT Press.

[263] C. Mead. Neuromorphic electronic systems. *Proceedings of the IEEE*, 78(10):1629–1636, October 1990.

[264] B. Mesot and C. Teuscher. Critical values in asynchronous random boolean networks. In W. Banzhaf, T. Christaller, P. Dittrich, J. T. Kim, and J. Ziegler, editors, *Advances in Artificial Life. Proceedings of the 7^{th} European Conference, ECAL2003*, volume 2801 of *Lecture Notes in Artificial Intelligence*, pages 367–376, Berlin, Heidelberg, 2003. Springer-Verlag.

[265] B. Mesot and C. Teuscher. Cellular automata versus random boolean networks. To be submitted to Physica D, 2004.

[266] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, Heidelberg, third edition, 1996.

[267] D. F. Miller, P. M. Todd, and S. U. Hegde. Designing neural networks using genetic algorithms. In J. D. Schaffer, editor, *Third International Conference on Genetic Algorithms*, pages 379–384. Morgan Kaufmann, 1989.

[268] J. F. Miller and K. Downing. Evolution *in materio*: Looking beyond the silicon box. In A. Stoica, J. Lohn, R. Katz, D. Keymeulen, and R. S. Zebulum, editors, *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware*, pages 167–176. IEEE Computer Society, Los Alamitos, CA, 2002.

[269] S. L. Miller. A production of amino acids under possible primitive earth conditions. *Science*, 117:528–529, May 15 1953.

[270] S. L. Miller and L. E. Orgel. *The Origins of Life on the Earth*. Prentice-Hall, Englewood Cliffs, NJ, 1974.

[271] S. L. Miller and H. C. Urey. Organic compound synthesis on the primitive earth. *Science*, 130(3370):245–251, July 31 1959.

[272] M. L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Englewood Cliffs, NJ, 1967.

[273] M. L. Minsky and S. Papert. *Perceptron: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, 1972.

[274] R. E. Mirollo and S. H. Strogatz. Synchronization of pulse-coupled biological oscillators. *SIAM Journal on Applied Mathematics*, 50(6):1645–1662, December 1990.

[275] M. Mitchell. *Analogy-Making as Perception: A Computer Model.* MIT Press, Cambridge, MA, 1993.

[276] M. Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, Cambridge, MA, 1996.

[277] T. Mitchell. *Machine Learning.* McGraw-Hill, New York, 1996.

[278] E. Mjolsness, D. H. Sharp, and J. Reinitz. A connectionist model of development. *Journal of Theoretical Biology*, 152:429–453, 1991.

[279] J. D. Murray. *Mathematical Biology.* Springer-Verlag, New York, 1993.

[280] K. Nagami, K. Oguri, T. Shiozawa, H. Ito, and R. Konishi. Plastic cell architecture: Towards reconfigurable computing for general-purpose. In K. L. Pocek and J. Arnold, editors, *Proceedings of the 6^{th} IEEE Symposium on FPGA-Based Custom Computing Machines (FCCM'98)*, pages 68–77, Los Alamitos, CA, 1998. IEEE Press.

[281] R. Nagpal. Organizing a global coordinate system from local information on an amorphous computer. Technical Report AI Memo 1666, MIT Artificial Intelligence Laboratory, August 12 1997.

[282] R. Nagpal. *Programmable Self-Assembly: Constructing Global Shape using Biologically-Inspired Local Interactions and Origami Mathematics.* PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, June 2001.

[283] R. Nagpal. *Programmable Self-Assembly: Constructing Global Shape using Biologically-inspired Local Interactions and Origami Mathematics.* PhD thesis, MIT Department of Electrical Engineering and Computer Science, June 2001.

[284] R. Nagpal and D. Coore. An algorithm for group formation in an amorphous computer. Technical Report AI Memo 1626, MIT Artificial Intelligence Laboratory, February 16 1998.

[285] S. Ohsuga nd H. Kangassalo, H. Jaakkola nd K. Hori, and N. Yonezaki. *Information Modeling and Knowledge Bases: Foundations, Theory, and Applications.* IOS Press, Amsterdam, 1991.

[286] S. B. Needleman and C. D.Wunsch. A general method applicable to the search for similarities in the amino acid sequences of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

[287] C. L. Nehaniv, editor. *Computation for Metaphor, Analogy and Agents*, volume 1562 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, Berlin, Heidelberg, 1999.

[288] C. L. Nehaniv. Evolution in asynchronous cellular automata. In Standish et al. [385], pages 65–73.

[289] A. Newell. Physical symbol systems. *Cognitive Science*, 4:135–183, 1980.

[290] A. Newell. The knowledge level. *Artificial Intelligence*, 18(87–127), 1982.

[291] A. Newell and H. Simon. *Human Problem Solving*. Prentice-Hall, Englewood Cliffs, NJ, 1972.

[292] J. Niittylahti. Hardware implementation of boolean neural network using simulated annealing. Master's thesis, Tampere University of Technology, Tampere, Finland, 1992.

[293] J. Niittylahti. *Boolean Neural Network Implementations*. PhD thesis, Tampere University of Technology, Tampere, Finland, 1995.

[294] J. Niittylahti. Hardware prototypes of a boolean neural network and the simulated annealing optimization method. *International Journal of Neural Systems*, 7(1):45–52, March 1996.

[295] J. Niittylahti. Boolean neural network trained with simulated annealing. In O. M. Omidvar, editor, *Progress in Neural Networks*, volume 6. Intellect Ltd, London, 1999.

[296] S. Nolfi and D. Floreano. Learning and evolution. *Autonomous Robots*, 7(1):89–113, 1999.

[297] S. Nolfi and D. Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press, Cambridge, MA, 2000.

[298] S. Nolfi, O. Miglio, and D. Parisi. Phenotypic plasticity in evolving neural networks. In P. Gaussier and J.-D. Nicoud, editors, *Proceedings of PerAc'94: From Perception to Action*, Los Alamitos, CA, 1994. IEEE Computer Society Press.

[299] S. Nolfi and D. Parisi. Growing neural networks. Technical Report PCIA-91-15, Institute of Psychology, CNR, Rome, 1991.

[300] S. Nolfi, D. Parisi, and J. L. Elman. Learning and evolution in neural networks. *Adaptive Behavior*, 3(1):5–28, 1994.

[301] M. A. Nowak, S. Bonhoeffer, and R. M. May. Spatial games and the maintenance of cooperation. *Proceedings of the National Academic of Sciences USA*, 91:4877–4881, May 1994.

[302] M. Nowostawski. Hierarchical code generators. In Bilotta et al. [44], pages 63–71.

[303] A. Obtulowicz and G. Paun. (In search of) probabilistic P systems. Available online: `http://psystems.disco.unimib.it/download/Probab.pdf`, October 2002.

[304] G. M. Odell, G. Oster, P. Albrech, and B. Burnside. The mechanical basis of morphogenesis I: Epithelial folding and invagination. *Developmental Biology*, 85:446–462, 1981.

[305] K. Oguri, N. Imlig, H. Ito, K. Nagami, R. Konishi, and T. Shiozawa. General purpose computer architecture based on fully programmable logic. In M. Sipper, D. Mange, and A. Pérez-Uribe, editors, *Proceedings of the Second International Conference on Evolvable Systems (ICES'98)*, volume 1478 of *Lecture Notes in Computer Science*, pages 324–334, Berlin, Germany, September 1998. Springer-Verlag.

[306] N. Ono. *Artificial Chemistry: Computational Studies on the Emergence of Self-Reproducting Units.* PhD thesis, Institute of Physics, University of Tokyo, March 16 2001.

[307] N. Ono and T. Ikegami. Self-maintenance and self-reproduction in an abstract cell model. *Journal of Theoretical Biology*, 206(2):243–253, September 21 2000.

[308] R. C. O'Reilly and Y. Munakata. *Computational Explorations in Cognitive Neuroscience.* A Bradford Book, MIT Press, Cambridge, MA, 2000.

[309] C. Ortega and A. Tyrrell. Self-repairable multicellular hardware: A reliability analysis. In D. Floreano, J.-D. Nicoud, and F. Mondada, editors, *Proceedings of the 5th European Conference on Artificial Life (ECAL'99)*, Advances in Artificial Life. Springer-Verlag, Berlin, 1999.

[310] C. A. Ortega-Sánchez. *Embryonics: A Bio-Inspired Fault-Tolerant Multicellular System.* PhD thesis, The University of York, Department of Electronics, May 2000.

[311] E. A. Di Paolo. Searching for rhythms in asynchronous boolean networks. In M. A. Bedau, J. S. McCaskill, N. H. Packard, and S. Rassmussen, editors, *Proceedings of the Seventh International Conference*

on Artificial Life, Reed College, Portland, OR, August 1–6 2000. A Bradford Book, MIT Press, Cambridge, MA; London, UK.

[312] E. A. Di Paolo. Rhythmic and non-rhythmic attractors in asynchronous random boolean networks. *Biosystems*, 59(3):185–195, 2001.

[313] Robert L. Park. *Voodoo Science*. Oxford University Press, New York, 2000.

[314] G. Paun. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000. First published in a TUCS Research Report, No 208, November 1998, `http://www.tucs.fi`.

[315] G. Paun. *Membrane Computing*. Springer-Verlag, Berlin, Heidelberg, Germany, 2002.

[316] G. Paun and G. Rozenberg. A guide to membrane computing. *Journal of Theoretical Computer Science*, 287(1):73–100, 2002.

[317] R. Penrose. *The Emperor's New Mind*. Oxford University Press, 1989.

[318] R. Penrose. *Shadows of the Mind*. Vintage, 1995.

[319] F. C. Pereira. Computational models of creativity. PhD thesis in preparation.

[320] F. C. Pereira and A. Cardoso. Knowledge integration with conceptual blending. In D. O'Donoghue, editor, *Proceedings of the Twelfth Irish Conference on Artificial Intelligence and Cognitive Science (AICS-2001)*, pages 33–42, Maynooth, Irland, September 2001. National University of Ireland, Department of Computer Science.

[321] F. C. Pereira and A. Cardoso. The boat-house visual blending experience. In *Proceedings of the European Conference on Artifical Intelligence (ECAI'02). Creative Systems: Approaches to Creativity in AI and Cognitive Science*, pages 71–77, Lyon, 2002.

[322] F. C. Pereira and A. Cardoso. Conceptual blending and the quest for the holy creative process. In *Proceedings of the European Conference on Artifical Intelligence (ECAI'02). Creative Systems: Approaches to Creativity in AI and Cognitive Science*, pages 67–70, Lyon, 2002.

[323] F. C. Pereira and A. Cardoso. The horse-bird creature generation experiment. *AISB Journal*, 2003.

[324] F. C. Pereira and A. Cardoso. Optimality principles for conceptual blending a first computational approach. In *Proceedings of the AISB'03 Symposium on Creativity in Arts and Sciences*, Aberystwyth. UK, April 2003.

[325] J. Piaget. *Logic and Psychology.* Manchester University Press, Manchester, UK, 1953.

[326] G. Pighizzini. Asynchronous automata versus asynchronous cellular automata. *Theoretical Computer Science*, 132:179–207, 1994.

[327] P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants.* Springer-Verlag, New York, 1990.

[328] Z. W. Pylyshyn. *Computation and Cognition.* MIT Press, Cambridge, MA, 1984.

[329] M. R. Quillian. Semantic memory. In M. Minsky, editor, *Semantic Information Processing*, chapter 4, pages 227–270. MIT Press, Cambridge, MA, 1968.

[330] N. Rambidi. Roots and promises of chemical-based computing. *BioSystems*, 64:169–178, 2002.

[331] M. Ramscar and D. Yarlett. Semantic grounding in models of analogy: An environmental approach. *Cognitive Science*, 27(1):41–71, January/February 2003.

[332] S. Rasmussen, N. A. Baas, B. Mayer, and M. Nilsson. Defense of the *Ansatz* for dynamical hierarchies. *Artificial Life*, 7(4):367–373, 2001.

[333] S. Rasmussen, N. A. Baas, B. Mayer, M. Nilsson, and M. W. Olesen. *Ansatz* for dynamical hierarchies. *Artificial Life*, 7(4):329–353, 2001.

[334] S. Rasmussen, L. Chen, M. Nilsson, and S. Abe. Bridging nonliving and living matter. *Artificial Life*, 9(3):269–316, 2003.

[335] S. Rasmussen, C. Knudsen, R. Feldberg, and M. Hindsholm. The Coreworld: emergence and evolution of cooperative structures in a computational chemistry. *Physica D*, 42:111–134, 1990.

[336] M. A. Reed and J. M. Tour. Computing with molecules. *Scientific American*, pages 69–75, June 2000.

[337] P. Rendell. Turing universality of the Game of Life. In A. Adamatzky, editor, *Collision-Based Computing*, pages 513–539. Springer-Verlag, London, 2002.

[338] J.-P. Rennard. Implementation of logical functions in the Game of Life. In A. Adamatzky, editor, *Collision-Based Computing*, pages 491–512. Springer-Verlag, London, 2002.

[339] J.-P. Rennard. *Vie Artificielle.* Vuibert, Paris, 2002.

[340] F. Rieke, D. Warland, R. de R. van Steveninck, and W. Bialek. *Spikes: Exploring the Neural Code.* A Bradford Book, MIT Press, Cambridge, MA; London, UK, second edition, 1996.

[341] S. Roberts and S. Turega. Evolving neural networks structures: An evaluation of encoding techniques. In D. W. Pearson, N. C. Steele, and R. F. Albrecht, editors, *Artificial Neural Nets and Genetic Algorithms. Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms, ICANNGA'95*, Wien, 1995. Springer-Verlag.

[342] M. C. Roco and W. S. Bainbridge, editors. *Converging Technologies for Improving Human Performance: Nanotechnology, Biotechnology, Information Technology and Cognitive Science.* World Technology Evaluation Center (WTEC), Arlington, Virginia, June 2002. NSF/DOC-sponsored report.

[343] R. Rojas. *Neural Networks: A Systematic Introduction.* Springer-Verlag, Berlin, 1996.

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

[345] E. M. A. Ronald, M. Sipper, and M. S. Capcarrère. Testing for emergence in artificial life. In D. Floreano, J.-D. Nicoud, and F. Mondada, editors, *Advances in Artificial Life: Proceedings of the 5th European Conference on Artificial Life (ECAL'99)*, volume 1674 of *Lecture Notes in Artificial Intelligence*, pages 13–20, Berlin, Heidelberg, 1999. Springer-Verlag.

[346] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

[347] F. Rosenblatt. *Principles of Neurodynamics.* Spartan, Washington, DC, 1961.

[348] L. I. Rozonoér. Random logical nets I. *Automation and Remote Control*, 5:773–781, 1969. Translation of Avtomatika i Telemekhanika.

[349] M. J. Russel, M. G. Switz, and K. Thompson. Olfactory influences on the human menstrual cycle. *Pharmacology Biochemistry and Behavior*, 13:737–738, 1980.

[350] S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, Upper Saddle River, New Jersey, 1995.

[351] A. G. Rust. *Developmental Self-Organisation in Artificial Neural Networks.* PhD thesis, University of Hertfordshire, July 1998.

[352] A. Samuel. Some studies in machine learning using the game of checkers. In E. Feigenbaum and J. Feldman, editors, *Computers and Thought.* McGraw-Hill, New York, 1963.

[353] E. Sanchez. An introduction to digital systems. In D. Mange and M. Tomassini, editors, *Bio-Inspired Computing Machines: Towards Novel Computational Architectures*, chapter 2, pages 13–47. Presses Polytechniques et Universitaires Romandes, Lausanne, Switzerland, 1998.

[354] E. Sanchez, D. Mange, M. Sipper, M. Tomassini, A. Pérez-Uribe, and A. Stauffer. Phylogeny, ontogeny, and epigenesis: Three source of biological inspiration for softening hardware. In *Proceedings of The First International Conference on Evolvable Systems: From Biology to Hardware (ICES96)*, volume 1259 of *Lecture Notes in Computer Science*, pages 35–54, Heidelberg, 1997. Springer-Verlag.

[355] P. T. Saunders, editor. *Collected Works of A. M. Turing: Morphogenesis.* North-Holland, Amsterdam, 1992.

[356] S. M. Scheiner. Genetics and evolution of phenotypic plasticity. *Annual Review of Ecological Systems*, 24:35–68, 1993.

[357] M. Scheuz, editor. *Computationalism: New Directions.* A Bradford Book, MIT Press, Cambridge, MA, 2002.

[358] B. Schönfisch and A. de Roos. Synchronous and asynchronous updating in cellular automata. *BioSystems*, 51(3):123–143, September 1999.

[359] E. Schroedinger. *What is Life?* University Press, Cambridge, 1967. First published in 1944.

[360] B. Schwichtenberg. Blending in action: Diagrams reveal conceptual integration in routine activity. *UCSD Cognitive Science Online*, 1:34–45, 2003. `http://cogsci-online.ucsd.edu/1/1-4.pdf`.

[361] A. C. Scott. *Nonlinear Science: Emergence and Dynamics of Coherent Structures.* Oxford University Press, Oxford, 1999.

[362] J. R. Searle. *The Rediscovery of the Mind.* MIT Press, Cambridge, MA, 1992.

[363] L. Sekanina. *Evolvable Components. From Theory to Hardware Implementations.* Springer-Verlag, Berlin, Heidelberg, 2004.

[364] O. G. Selfridge. "Pandemonium": A paradigm for learning. In *Mechanisation of Thought Processes: Proceedings of a Symposium held at the National Physical Laboratory*, pages 513–526, 1958.

[365] O. G. Selfridge and U. Neisser. Pattern recognition by machine. *Scientific American*, 203(2):60–68, 1960.

[366] J. M. Seminario, L. E. Córdova, and P. A. Derosa. An *ab initio* approach to the calulation of current-voltage characteristics of programmable molecular devices. *Proceedings of the IEEE*, 91(11):1958–1975, 2003.

[367] H. T. Siegelmann. Computation beyond the Turing limit. *Science*, 268(5210):545–548, April 1995.

[368] H. T. Siegelmann. The simple dynamics of super Turing theories. *Theoretical Computer Science*, 168:461–472, 1996.

[369] H. T. Siegelmann. *Neural Networks and Analog Computation: Beyond the Turing Limit*. Progress in Theoretical Computer Science. Birkhauser Verlag, November 1998.

[370] S. N. Simic and S. Sastry. Distributed environmental monitoring using random sensor networks. In F. Zhao and L. Guibas, editors, *Information Processing in Sensor Networks (IPSN2003)*, volume 2634 of *Lecture Notes in Computer Science*, pages 582–592. Springer-Verlag Berlin Heidelberg, 2003.

[371] M. Sipper. *Evolution of Parallel Cellular Machines: The Cellular Programming Approach*. Springer-Verlag, Heidelberg, 1997.

[372] M. Sipper. The emergence of cellular computing. *IEEE Computer*, 32(7):18–26, July 1999.

[373] M. Sipper. *Machine Nature: The Coming Age of Bio-Inspired Computing*. McGraw-Hill, New York, 2002.

[374] M. Sipper, M. Goeke, D. Mange, A. Stauffer, E. Sanchez, and M. Tomassini. The firefly machine: Online evolware. In *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation (ICEC'97)*, pages 181–186. IEEE, 1997.

[375] M. Sipper and E. Ruppin. Co-evolving architectures for cellular machines. *Physica D*, 99:428–441, 1997.

[376] M. Sipper, E. Sanchez, D. Mange, M. Tomassini, A. Pérez-Uribe, and A. Stauffer. A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems. *IEEE Transactions on Evolutionary Computation*, 1(1):83–97, April 1997.

[377] M. Sipper, E. Sanchez, D. Mange, M. Tomassini, A. Pérez-Uribe, and A. Stauffer. An introduction to bio-inspired machines. In D. Mange and M. Tomassini, editors, *Bio-Inspired Computing Machines: Towards Novel Computational Architectures*, chapter 1, pages 1–12. Presses Polytechniques et Universitaires Romandes, Lausanne, Switzerland, 1998.

[378] M. Sipper, M. Tomassini, and O. Beuret. Studying probabilistic faults in evolved non-uniform cellular automata. *International Journal of Modern Physics*, 7(6):923–939, 1996.

[379] M. Sipper, M. Tomassini, and M. S. Capcarrere. Evolving asynchronous and scalable non-uniform cellular automata. In G. D. Smith, N. C. Steele, and R. F. Albrecht, editors, *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms (ICANNGA97)*, pages 66–70, Wien, 1997. Springer-Verlag.

[380] A. Sloman. The irrelevance of turing machines to artificial intelligence. In Scheuz [357], pages 87–127.

[381] D. R. Smith and C. H. Davidson. Maintained activity in neural nets. *Journal of the ACM*, 9:268–279, 1962.

[382] E. E. Smith, D. N. Osherson, L. J. Rips, and M. Keane. Combining prototypes: A selective modification model. In Margolis and Laurence [249], chapter 17, pages 355–390.

[383] P. Smolensky. On the propoer treatment of connectionism. *Behavioral and Brain Sciences*, 11(1):1–74, 1988.

[384] M. R. Stan, P. D. Franzon, S. C. Goldstein, J. C. Lach, and M. M. Ziegler. Molecular electronics: From devices and interconnect to circuits and architectures. *Proceedings of the IEEE*, 91(11):1940–1957, 2003.

[385] R. K. Standish, M. A. Bedau, and H. A. Abbass, editors. *Artificial Life VIII. Proceedings of the Eight International Conference on Artificial Life*. Complex Adaptive Systems Series. A Bradford Book, MIT Press, Cambridge, MA, 2003.

[386] M. Stannett. X-machines and the halting problem: Building a super-Turing machine. *Formal Aspects of Computing*, 2(4):331–341, 1990.

[387] R. W. Stark. Dynamics for fundamental problem of biological information processing. *International Journal of Artificial Intelligence Tools*, 4(4):471–488, 1995.

[388] W. R. Stark and W. H. Hughes. Asynchronous, irregular automata nets: The path not taken. *BioSystems*, 55(1-3):107–117, February 2000.

[389] A. Stauffer, D. Mange, G. Tempesti, and C. Teuscher. BioWatch: A giant electronic bio-inspired watch. In D. Keymeulen, A. Stoica, J. Lohn, and R. S. Zebulum, editors, *Proceedings of the Third NASA/DoD Workshop on Evolvable Hardware, EH-2001*, pages 185–192, Los Alamitos, CA, 2001. IEEE Computer Society.

[390] A. Stauffer, D. Mange, G. Tempesti, and C. Teuscher. A self-repairing and self-healing electronic watch: The BioWatch. In Y. Liu, K. Tanaka, M. Iwata, T. Higuchi, and M. Yasunaga, editors, *Evolvable Systems: From Biology to Hardware. Proceedings of the 4th International Conference on Evolvable Systems, ICES2001, Tokyo, October 3-5, 2001*, volume 2210 of *Lecture Notes in Computer Science*, pages 112–127, Berlin, Heidelberg, 2001. Springer-Verlag.

[391] O. Steinbock, P. Kettunen, and K. Showalter. Chemical wave logic gates. *J. Phys. Chem.*, 100:18970–18975, 1996.

[392] O. Steinbock, A. Toth, and K. Showalter. Navigating complex labyrinths: Optimal paths from chemical waves. *Science*, 267(5199):868–871, 1995.

[393] I. Stewart. Deciding the undecidable. *Nature*, 352:664–665, 1991.

[394] S. H. Strogatz and I. Stewart. Coupled oscillators and biological synchronization. *Scientific American*, 269:68–75, December 1983.

[395] M. P. Stryker. Precise development from imprecise rules. *Science*, 263:1244–1245, 1994.

[396] L. A. Suchman. *Plans and Situated Actions.* Cambridge University Press, Cambridge, 1987.

[397] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* A Bradford Book, MIT Press, Cambridge, MA, 1998.

[398] Y. Suzuki and H. Tanaka. Chemical evolution among artificial protocells. In Bedau et al. [33], pages 54–63.

[399] Y. Suzuki and H. Tanaka. A new molecular computing model, artificial cell system. In H. Beyer, E. Cantu-Paz, D. Goldberg, I. Parmee, L. Spector, and D. Whitley, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO2000)*, pages 833–840, San Francisco, CA, 2000. Morgan Kaufmann Publishers, Inc.

[400] A. Syropoulos, E. G. Mamatas, P. C. Allilomes, and K. T. Sotiriades. A distributed simulation of P systems. In A. Alhazov, C. Martín-Vide, and G. Paun, editors, *Proceedings of the MolCoNet Workshop on Membrane Computing (WMC2003)*, volume 28/03, pages 455–460, Tarragona (Spain), 2003. Rovira i Virgili University, Research Group on Mathematical Linguistics.

[401] A. S. Tanenbaum, editor. *Computer Networks.* Prentice Hall, Upper Saddle River, NJ, 4 edition, 2003.

[402] U. Tangen, L. Schulte, and J. S. McCaskill. A parallel hardware evolvable computer POLYP. In K. L. Pocek and J. Arnold, editors, *Proceedings of the 5^{th} IEEE Symposium on FPGA-Based Custom Computing Machines (FCCM'97)*, pages 238–239, Los Alamitos, CA, 1997. IEEE Press.

[403] G. Tempesti. *A Self-Repairing Multiplexer-Based FPGA Inspired by Biological Processes.* PhD thesis, Swiss Federal Institute of Technology Lausanne, CH-1015 Lausanne, 1998. Thesis No 1827.

[404] G. Tempesti, D. Mange, A. Stauffer, and C. Teuscher. The Biowall: An electronic tissue for prototyping bio-inspired systems. In A. Stoica, J. Lohn, R. Katz, D. Keymeulen, and R. S. Zebulum, editors, *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware*, pages 221–230, Los Alamitos, CA, 2002. IEEE Computer Society.

[405] G. Tempesti, D. Roggen, E. Sanchez, Y. Thoma, R. Canham, A. Tyrrell, and J.-M. Moreno. A POEtic architecture for bio-inspired hardware. In Standish et al. [385].

[406] G. Tempesti and C. Teuscher. Biology goes digital: An array of 5700 Spartan FPGAs brings the BioWall to "life.". *Xilinx XCell*, 47:40–45, Fall 2003.

[407] A. Tettamanzi and M. Tomassini. *Soft Computing: Integrating Evolutionary, Neural, and Fuzzy Systems.* Springer-Verlag, Berlin, Heiderlberg, 2001.

[408] C. Teuscher. *Turing's Connectionism. An Investigation of Neural Network Architectures.* Springer-Verlag, London, September 2002.

[409] C. Teuscher, editor. *Alan Turing: Life and Legacy of a Great Thinker.* Springer-Verlag, Berlin, Heidelberg, 2004.

[410] C. Teuscher. Turing's connectionism. In C. Teuscher, editor, *Alan Turing: Life and Legacy of a Great Thinker*, pages 499–530. Springer-Verlag, Berlin, Heidelberg, 2004.

[411] C. Teuscher and M. S. Capcarrère. On fireflies, cellular systems, and evolware. In A. M. Tyrrell, P. C. Haddow, and J. Torresen, editors, *Evolvable Systems: From Biology to Hardware. Proceedings of the* 5^{th} *International Conference (ICES2003)*, volume 2606 of *Lecture Notes in Computer Science*, pages 1–12, Berlin, Heidelberg, 2003. Springer-Verlag.

[412] C. Teuscher, D. Mange, A. Stauffer, and G. Tempesti. Bio-inspired computing tissues: Towards machines that evolve, grow, and learn. *BioSystems*, 68(2–3):235–244, February–March 2003.

[413] C. Teuscher and E. Sánchez. Self-organizing topology evolution of Turing neural networks. In G. Dorffner, H. Bischof, and K. Hornik, editors, *Proceedings of the International Conference on Artificial Neural Networks (ICANN2001)*, volume 2130 of *Lecture Notes in Computer Science*, pages 820–826, Berlin, Heidelberg, 2001. Springer-Verlag.

[414] D. W. Thompson. *On Growth and Form.* Cambridge University Press, Cambridge, UK, 1961.

[415] R. L. Thompson and N. S. Goel. Movable Finite Automata (MFA) models for biological systems I: Bacteriophage assembly and operation. *Journal of Theoretical Biology*, 131:351–385, 1988.

[416] T. Toffoli and N. Margolus. *Cellular Automata Machines.* MIT Press, Cambridge, MA, 1987.

[417] T. Toffoli and N. Margolus. Programmable matter: Concepts and realizations. *Physica D*, 47:263–272, 1991.

[418] M. Tomassini and M. Venzi. Evolving robust asynchronous cellular automata for the density task. *Complex Systems*, 13:185–204, 2002.

[419] M. Tomita. Whole-cell simulation: A grand challenge of the 21st century. *Trends in Biotechnology*, 19(6):205–210, June 1 2001.

[420] B. A. Trakhtenbrot. Comparing the Church and Turing approaches: Two prophetical messages. In Herken [188], pages 557–582.

[421] S. M. Trimberger. *Field-Programmable Gate Array Technology.* Kluwer Academic Publishers, Boston, 1994.

[422] A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. In *Proceedings of the London Mathematical Society*, volume 42 of *2*, pages 230–265, 1936-7. Corrections in *Proceedings of the London Mathematical Society*, volume 43, pages 544–546, 1937.

[423] A. M. Turing. Systems of logic based on ordinals. In *Proceedings of the London Mathematical Society*, volume 45 of *2*, pages 161–228, 1939. This was also Turing's Princeton Ph.D. thesis (1938).

[424] A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

[425] A. M. Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London*, B 237:37–72, 1952.

[426] A. M. Turing. Intelligent machinery. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 5, pages 3–23. Edinburgh University Press, Edinburgh, 1969.

[427] M. Turner. *Cognitive Dimensions of Social Science*. Oxford University Press, New York, NY, 2001.

[428] A. Tyrrell, E. Sanchez, D. Floreano, G. Tempesti, D. Mange, J.-M. Moreno, J. Rosenberg, and Alessandro E. P. Villa. Poetic tissue: An integrated architecture for bio-inspired hardware. In A. M. Tyrrell, P. C. Haddow, and J. Torresen, editors, *Evolvable Systems: From Biology to Hardware. Proceedings of the* 5^{th} *International Conference (ICES2003)*, volume 2606 of *Lecture Notes in Computer Science*, pages 129–140. Springer-Verlag, Berlin, Heidelberg, 2003.

[429] H. R. Ueda, K. Hirose, and M. Iino. Intercellular coupling mechanism for synchronized and noise-resistant circadian oscillators. *Journal of Theoretical Biology*, 216:501–512, 2002.

[430] P. Ulmschneider. *Intelligent Life in the Universe: From Common Origins to the Future of Humanity*. Springer-Verlag, 2003.

[431] J. Vaario. *An Emergent Modeling Method for Artificial Neural Networks*. PhD thesis, The University of Tokyo, 1993.

[432] J. Vaario, S. Ohsuga, and K. Hori. Connectionist modeling using Lindenmayer systems. In *Information Modeling and Knowledge Bases: Foundations, Theory, and Applications* [285], pages 496–510.

[433] Jari Vaario and Setsuo Ohsuga. On growing intelligence. In Dorffner [113].

[434] F. Varela. *Principles of Biological Autonomy*, volume 2 of *Series in General Systems Research*. North Holland, New York, 1979.

[435] F. Varela, H. Maturana, and R. Uribe. Autopoiesis: The organization of living systems, its characterization and a model. *BioSystems*, 5:187–196, 1974.

[436] T. Veale. *Metaphor, Memory and Meaning: Symbolic and Connectionist Issues in Metaphor Comprehension.* PhD thesis, Trinity College, Doublin, Ireland, 1995.

[437] T. Veale and D. O'Donoghue. Computation and blending. *Cognitive Linguistics*, 11(3–4):253–281, 2000.

[438] T. Veale, D. O'Donoghue, and M. T. Keane. Epistemological issues in metaphor comprehension: A comparison of three models and a new theory of metaphor. In *Proceedings of the International Cognitive Linguistics Conference (ICLA'95)*, University of New Mexico, Albuquerque, July 17–21 1995.

[439] A. H. Veen. Dataflow machine architecture. *ACM Computing Surveys*, 37(4):365–396, December 1986.

[440] P. F. M. J. Verschure and P. Althaus. A real-world rational agent: Unifying old and new AI. *Cognitive Science*, 27(4):561–590, 2003.

[441] J. Villasenor and W. H. Mangione-Smith. Configurable computing. *Scientific American*, 276(6):54–59, June 1997.

[442] Ch. von der Malsburg. Self-organization and the brain. In Arbib [13].

[443] J. von Neumann. First draft of a report on the EDVAC. Technical report, Moore School of Electrical Engineering, University of Pennsylvania, June 30 1945.

[444] J. von Neumann. Probabilistic logic and the synthesis of reliable organisms from unreliable components. In *Automata Studies*, pages 43–98. Princeton University Press, Princeton, NJ, 1956.

[445] J. von Neumann. *Theory of Self-Reproducing Automata.* University of Illinois Press, Urbana, Illinois, 1966.

[446] M. D. Vose. *The Simple Genetic Algorithm: Foundations and Theory.* MIT Press, Cambridge, MA, 1999.

[447] J. F. Wakerly. *Digital Design: Principles & Practices.* Prentice Hall International Inc., New Jersey, third edition, 2000.

[448] C. C. Walker. Behavior of a class of complex systems: the effect of system size on properties of terminal cycles. *Journal of Cybernetics*, 1(4):55–67, 1971.

[449] C. C. Walker. Attractor dominance patterns in sparsely connected boolean nets. *Physica D*, 45:441–451, 1990.

[450] T. J. Walker. Acoustic synchrony: Two mechanisms in the snowy tree cricket. *Science*, 166:891–894, 1969.

[451] B. Warneke, M. Last, B. Liebowitz, and K. S. J. Pister. Smart dust: Communication with a cubic-millimeter computer. *IEEE Computer*, 34(1):44–51, January 2001.

[452] R. A. Watson. Modular independance in complex dynamical systems. In Bilotta et al. [44], pages 81–88.

[453] P. Wegner. Interactive foundations of computing. *Theorectical Computer Science*, 192:315–351, 1998.

[454] G. Weisbuch. *Dynamique des systèmes complexes: Une introduction aux réseaux d'automates.* InterEditions, France, 1989.

[455] G. Weisbuch. *Complex Systems Dynamics: An Introduction to Automata Networks*, volume 2 of *Lecture Notes, Santa Fe Institute, Studies in the Sciences of Complexity.* Addison-Wesley, Redwood City, CA, 1991.

[456] R. Weiss. *Cellular Computation and Communications using Engineered Genetic Regulatory Networks.* PhD thesis, MIT, Department of Electrical Engineering and Computer Science, September 2001.

[457] M. Weliky and G. Oster. The mechanical basis of cell rearrangement. 1: Epithelial morphogenesis during fundulus epiboly. *Development*, 109:373–386, 1990.

[458] S. Wermter, J. Austin, and D. Willshaw, editors. *Emergent Neural Computation Architectures Based on Neuroscience. Towards Neuroscience-Inspired Computing.* Number 2036 in Lecture Notes in Artificial Intelligence (LNAI). Springer-Verlag, Berlin, Heidelberg, 2001.

[459] S. Wermter, S. Austin, D. Willshaw, and M. Elshaw. Towards novel neuroscience-inspired computing. In Wermter et al. [458], pages 1–19.

[460] N. Wiener. *Cybernetics or Control and Communication in the Animal and Machine.* MIT Press, Cambridge, MA, 1948.

[461] K. Wiesenfeld and F. Jaramillo. Minireview of stochastic resonance. *Chaos*, 8(3):539–548, 1998.

[462] K. Wiesenfeld and F. Moss. Stochastic resonance and the benefits of noise: From ice ages to crayfish and SQUIDs. *Nature*, 373:33–36, 1995.

[463] S. Wolfram. Cellular automata as models of complexity. *Nature*, 311:419–424, December 4 1984.

[464] S. Wolfram. Approaches to complexity engineering. *Physica D*, 22:385–399, 1986.

[465] S. Wolfram. *A New Kind of Science*. Wolfram Media, Inc., 2002.

[466] L. Wolpert. Positional information and the spatial pattern of cellular differentiation. *Journal of Theoretical Biology*, 25(1):1–47, October 1969.

[467] S. Ramón y Cajal. *Advice for a Young Investigator*. MIT Press, Cambridge, MA, 1999.

[468] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, September 1999.

[469] X. Yao and Y. Liu. Evolving artificial neural networks through evolutionary programming. In *Fifth Annual Conference on Evolutionary Programming*, pages 257–266, San Diego, CA, 2 March 1996. MIT Press.

[470] X. Yao and Y. Liu. A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, 8(3):694–713, 1997.

[471] J. Zahnd. *Logique élémentaire*. Presses Polytechniques et Universitaires Romandes, 1998.

[472] K. P. Zauner and M. Conrad. Parallel computing with DNA: Toward the anti-universal machine. In *Parallel Problem Solving from Nature (PPSN IV)*, volume 1141 of *Lecture Notes in Computer Science*, pages 696–705, Berlin, Heiderlberg, 1996. Springer-Verlag.

[473] B. P. Zeigler. The brain-machine disanalogy revisited. *BioSystems*, 64(1–3):127–140, January 2002.

[474] V. V. Zhirnov and D. J. C. Herr. New frontiers: Self-assembly in nanoelectronics. *IEEE Computer*, pages 34–43, January 2001.

[475] C. Zhou and J. Kurths. Noise-induced phase synchronization and synchronization transitions in chaotic oscillators. *Physical Review Letters*, 88(23), 2002.

[476] W. Zielonka. Notes on finite asynchronous automata. *Informatique théorique et Applications/Theoretical Infomatics and Applications*, 21(2):99–135, 1987.

[477] W. Zielonka. Safe executions of recognizable trace languages by asynchronous automata. In Albert R. Meyer and Michael A. Taitslin, editors, *Logic at Botik'89*, Lecture Notes in Computer Science, LNCS 363, pages 278–289, Berlin, 1989. Springer-Verlag.

Curriculum Vitae

Reality continues to ruin my life.

Calvin and Hobbes

Personal Data

Date of birth	July 27, 1972
Place of birth	Erlenbach, Switzerland
Martial status	Married to Ursina Teuscher-Wüthrich
Citizenship	Swiss
Title	ingénieur informaticien dipl. EPFL (eq. to a MSCS degree), dipl. Elektroniker (electronic technician)

Professional Experience, Career Path, and Highlights

2004–	**Postdoctoral Researcher** University of California San Diego (UCSD), Cognitive Science Department, Complex Systems & Cognition Laboratory.
2000–2004	**UNIX System Administrator** System administrator for about 20 SUN workstations,

several Linux PCs, a HP server and a SUN Enterprise server.

2000–2004 **Research assistant and PhD candidate**
Swiss Federal Institute of Technology Lausanne (EPFL), Logic Systems Laboratory, School of Computer and Communication Sciences.

1998–2000 **Internship** (part-time) and own **Undergraduate Student Research Projects**
Swiss Federal Institute of Technology Lausanne (EPFL), Logic Systems Laboratory, Computer Science Department. Projects done as a student and during an internship:

- LCD/LED display evaluation for the BioWall project. Special envoy to a display fair in Hong Kong.
- Study, implementation, and evolution of neural networks proposed by A. M. Turing. Results in the publication of a book by Springer-Verlag.
- Implementation of a cryptographic co-processor using the IDEA algorithm. Industrial research project in collaboration with *Lightning Ltd.*, Lausanne, Switzerland.
- Implementation of a 32-bit VLIW low power processor.
- Development of an autonomous robot and application of evolutionary algorithms. Results in the publication of several scientific papers.
- Development and test of the reconfigurable platform *Labomat3* (Xilinx FPGAs, Motorola 68360, RTEMS OS). Results in the publication of several scientific papers.

1998–2004 **Siemens International Student Program**
Trainee program for highly qualified, internationally oriented junior executives. Participation in several management and teamwork seminars in Germany.

1996–2000 **Intelligent Battery Systems (IBS), Thun, Switzerland**
Part-time work:

- Development of a multi-tasking software (80C517) for a battery-diagnosis prototyping system.
- Design and maintenance of the IBS web-site.

- Development of an adaptive, genetic algorithm based diagnosis system for car batteries. Project presentation at *Audi AG*, Ingolstadt, Germany.

1995–1997 **Theater *"Das Kleine Freudenhaus"*, Thun, Switzerland**
Development and implementation of two complex, PC-based, real-time control systems with several hundred I/Os.

1991–1995 **Start-up**
Formation of the company AIOLOtronic, Erlenbach, Switzerland.

1990–1993 **Development and Commercialization of the Wind-Measuring Computer *Aiolos II***
Participation in the *Swiss Contest for Young Scientists* and the 3^{rd} *European Community Contest for Young Scientists*. Commercialization and selling of the system with own company AIOLOtronic.

Education

2000–2004 **PhD Thesis. Logic Systems Laboratory, Swiss Federal Institute of Technology Lausanne (EPFL).**
Amorphous Membrane Blending: From Regular to Irregular Cellular Computing Machines. Thesis supervisor: Prof. Daniel Mange.

1995–2000 **Swiss Federal Institute of Technology Lausanne (EPFL).**
Computer Science Department. Diploma in Computer Science (equivalent to a MSCS degree). Graduated with several honors in 2000. Diploma thesis: *"Study, Implementation, and Evolution of the Neural Networks Proposed by Alan M. Turing"*.

1992–1995 **College, Thun, Switzerland.**
Graduated in 1995.

1989–1992 **Technical High School, Bern, Switzerland.**
Graduated in 1992.

1988–1992 **Technical College, Bern, Switzerland.**
Graduated in 1992.

1988–1992 **Elektronikbetrieb Zweisimmen, Switzerland.**
Apprenticeship as Electrician. Graduated with honors in 1992.

Special Career Highlights

2003 **Guest Editor**
C. Teuscher. *BioSystems Special Issue: IPCAT2003.* To be published in 2004.

2003 **Book Publication**
C. Teuscher (ed.). *Alan Turing: Life and Legacy of a Great Thinker.* Springer-Verlag, Berlin, Heidelberg, 2004.
(See also: `http://www.teuscher.ch/alanturing`)

2003 Nominated for a fellowship at the *Center for Advanced Study in the Behavioral Sciences* at Stanford University.

2001 **Book Publication**
C. Teuscher. *Turing's Connectionism. An Investigation of Neural Network Architectures.* Springer-Verlag, London, 2002.
(See also: `http://www.teuscher.ch/tc`)

2000– **Head of the BioWall Project**
Responsible for a budget of more than $350'000. Collaboration with *Villa Reuge Museum*, Ste-Croix, Switzerland. The project was widely covered by the media.

2001 **European Community Research Project: POEtic**
Active participation in the definition and the proposal-writing of a successful European Community research project (IST IST-2000-28027). Partners: EPFL, University of Lausanne, University of Catalunya, University of York, University of Glasgow. (See also: `http://www.poetictissue.org`)

2001 **Patent**
Person in charge of a European patent proposal

(No 01201221.7) related to the BioWall project. Co-authored with D. Mange, G. Tempesti, and A. Stauffer.

Conference Organization

2003 **IPCAT2003**
Organizer and program-chair of the 5^{th} International Conference on Information Processing in Cells and Tissues (IPCAT2003), September 8–11, 2003, Lausanne, Switzerland.

2002 **Turing Day**
Initiator, organizer and general chair of the *Turing Day*, an international workshop to commemorate the 90^{th} anniversary of Alan Mathison Turing's birthday. Invited talk: *Connectionism, Turing, and the Brain.* A commemorative festschrift was published by Springer-Verlag in 2004. Editor: Christof Teuscher. (See also: `http://www.teuscher.ch/alanturing`)

1999 **Genetic and Evolutionary Computation Conference, GECCO'99**
Orlando, Florida, USA, July 13-17, 1999.
Session chairman, PhD Workshop.

Research Interests

- Biologically-inspired computation and machines
- Novel hardware and reconfigurable architectures
- Amorphous computing and membrane computing
- Unconventional models of computation
- Machine intelligence, AI, connectionism
- Fault-tolerance
- Computation in uncertain environments
- Complex dynamic systems and chaos
- The science of networks
- Cellular systems and automaton
- Cognitive science and psychology
- Computational modeling of cognitive phenomena

Research Activities

2002– **Face2Face:** An internet study investigating attractiveness judgments depending on the age of the faces presented. A central question is whether the sexual orientation of the subjects makes a difference in their rating patterns, for instance, if female faces of varying age are rated in a different way by lesbians and heterosexual men. With Ursina Teuscher-Wüthrich.

2001– **Firefly and Density Task:** Investigating synchronous and asynchronous firefly synchronization and density task systems based on regular, irregular, and random structures.

2002–2003 **Alan M. Turing: Life and Legacy of a Great Thinker**. Editor of a commemorative festschrift (about 600 pages, more than 20 contributions) published by Springer-Verlag, 2004.

2000–2004 **PhD Thesis**: *Amorphous Membrane Blending.* Development of novel and unconventional biologically-inspired computing machines and tissues inspired by *amorphous computing*, *membrane computing*, *artificial chemistries*, and *conceptual integration (blending)*.

2000–2003 **BioWall:** Head of the *BioWall* project. Development of the BioWall reconfigurable computing tissue. With D. Mange, A. Stauffer, G. Tempesti, F. Vannel, Y. Thoma. Collaboration with *Villa Reuge Museum*, Ste-Croix, Switzerland.

2000– **Random Boolean Networks:** Investigating the complex dynamics of asynchronous and synchronous random boolean networks and their various variants versus cellular automata.

1999–2001 **Turing's Connectionism:** Investigation of the neural networks proposed by Alan M. Turing. Author of the book *Turing's Connectionism*, Springer-Verlag, 2002.

1998–2000 **CryptoBooster:** Development of a reconfigurable and modular cryptographic coprocessor. Industrial research project (Lightning Ltd., Lausanne, Switzerland). With E. Sanchez, E. Mosanya, J.-L. Beuchat, F. Gomez.

1997–1999 **Labomat3:** Development of a reconfigurable (FPGA) platform for academic purposes. With Jacques-Olivier Haenni and others.

1990–1993 **Development and commercialization of the Wind -Measuring Computer *Aiolos II*.** With Flavio Stragiotti.

Awards, Honors, and Grants

2000 **Asea Brown Boveri Award 2000**
Received for the diploma thesis entitled *"Study, Implementation, and Evolution of the Neural Networks Proposed by Alan M. Turing"*. The prize is intended to reward an important contribution in the field of computer science and telecommunications.

2000 **Jean Landry Award**
Received for the diploma thesis entitled *"Study, Implementation, and Evolution of the Neural Networks Proposed by Alan M. Turing"*. The prize is intended to reward an original and personal scientific work.

2000 **Annaheim Foundation Award**
Received for the diploma thesis entitled *"Study, Implementation, and Evolution of the Neural Networks Proposed by Alan M. Turing"*. The prize is intended to reward an excellent work in the domain of biologically-inspired systems.

2000 **Swiss Informaticians Society Award**
Awards the second best diploma.

1999 **Student Travel Grant Award.**
Genetic and Evolutionary Computation Conference, GECCO'99, Orlando, Florida, USA.

1992 **Award for final examination**
Electrician (dipl. Elektroniker).

1991 1^{st} **Prize, Swiss Contest for Young Scientists.**
Award for the project *Aiolos II* and nomination for the 3^{rd} *European Community Contest for Young Scientists.* HEUREKA science fair, Zürich.

1991 — 2^{nd} **Prize, 3^{rd} European Community Contest for Young Scientists**
Award for the project *Aiolos II.* HEUREKA science fair, Zürich.

1986 — 1^{st} **Prize**, musical contest **"Musikwettspiele Biel-Mett"**, clarinet.

1985 — 3^{rd} **Prize**, musical contest **"Musikwettspiele Frutigen"**, clarinet.

1984 — 2^{nd} **Prize**, musical contest **"Musikwettspiele Boezingen"**, clarinet.

Teaching Activities

Courses:

2002 — **Course "Künstliches Leben—eine spielerische Entdeckungsreise"** (artificial life course), Playground Summer School, Gottlieb Duttweiler Institut, Zürich.

Interventions:

2001–2004 — **Course "Systèmes et programmation génétiques"** (biologically-inspired systems and methods), guest lecturer, Swiss Federal Institute of Technology Lausanne (EPFL), Switzerland.

2003–2004 — **Course "Antropologie culturelle et sociale: Initiation à la recherche empirique"** (cultural and social anthropology), guest lecturer, University of Lausanne, Switzerland.

Teaching Assistant, Swiss Federal Inst. of Techn., Lausanne:

1998 — Programming in LISP, Prolog, Miranda. Prof. Charles Rapin

1998–2001 — Processor architecture. Prof. Eduardo Sanchez

1998–2001 — Computer architecture. Prof. Eduardo Sanchez

1999–2001 — Advanced digital systems design. Prof. Eduardo Sanchez

1998–2001	Digital systems design. Prof. Daniel Mange
2000–2001	Digital systems design. Prof. Eduardo Sanchez
2000–2004	Elementary logic. Prof. Jacques Zahnd
2001–2004	Automata and computation. Prof. Jacques Zahnd

Invited Talks and Selected Presentations

2004	**Le Mont Saint Michel, France**. Workshop on Unconventional Programming Paradigms (UPP). Talk: *Amorphous Membrane Computing.*
2004	**Université d'Every, France**, Workshop on the Modelling and Simulation of Biological Processes in the Context of Genomics. Talk: *From Nature to Machines and Back?*
2002	**EPFL, Turing Day**. Talk: *Connectionism, Turing, and the Brain.*
2003	**HP Labs, Palo Alto, CA**. Talk: *From Embryonics to the BioWall.*
2003	**Stanford University, CA**. Talk: *Biologically-Inspired Computing: Quo Vadis?*
2003	**NASA JPL, Pasadena, CA**. Talk: *Biologically-Inspired Computing: Quo Vadis?*

Languages

German	Native speaker
Swiss-German	Native speaker
French	Excellent
English	Excellent

Military Service

1998–	KDO SKS KDO GLG, EDV Pi (computer science army, general staff command), Luzern, Switzerland.
1996	EDV Kp 1/47, EDV Pi (computer science army, general staff).
1992	Geb Inf RS211, (mountain soldier), Andermatt, Switzerland.

Professional Organization Memberships

ALIFE	International Society for Artificial Life (ISAL)
ENNS	European Neural Network Society
Eunite	European Network on Intelligent Technologies for Smart Adaptive Systems
EvoWeb	European Network of Excellence in Evolutionary Computing
SI	Swiss Informaticians Society
SVI	Swiss Federation of Information Processing Societies
ACM	Association for Computing Machinery
IEEE	Institute of Electrical and Electronics Engineers
IEEE CS	IEEE Computer Society
SISP	Siemens International Student Program
GJ	Alumni Association of Young Scientists

Boards and Program Committees

EH-2004	**Program Committee:** NASA/DoD Conference on Evolvable Hardware (EH-2004).
Bio-ADIT2004	**Publicity Chair:** The First International Workshop on Biologically Inspired Approaches to Advanced Information Technology, January 29–30, 2004, EPFL, Lausanne.

IPCAT2003 **Organizer and Program-Chair:** 5^{th} International Workshop on Information Processing in Cells and Tissues (IPCAT2003), September 8–11, 2003, Lausanne, Switzerland.

EH-2003 **Program Committee:** NASA/DoD Conference on Evolvable Hardware (EH-2003), Chicago, IL.

ECAL2003 **Program Committee:** 7^{th} European Conference on Artificial Life (ECAL2003), Dortmund, Germany.

ICES2003 **Program Committee:** 5^{th} International Conference on Evolvable Systems: From Biology to Hardware (ICES2003), Trondheim, Norway.

Turing Day **Initiator, Organizer, and General Chair:** Turing Day, an international workshop to commemorate the 90^{th} anniversary of Alan Mathison Turing's birthday, June 28, 2002, Lausanne, Switzerland.
(See also: `http://www.teuscher.ch/turingday`)

2002–2004 **Member, "Conseil de moyens informatiques"** (board of the faculty computing resources), School of Computer and Communication Science, Swiss Federal Institute of Technology Lausanne (EPFL).

2000–2002 **Member, "Conseil de moyens informatiques"** (board of the faculty computing resources), Computer Science Department, Swiss Federal Institute of Technology Lausanne (EPFL).

2000–2002 **Member, Board of the Computer Science Department**, Swiss Federal Institute of Technology Lausanne (EPFL).

Occasional Reviewer

2003– **Springer-Verlag, London.**

2002– **Journal of Genetic Programming and Evolvable Machines.**

2002– **Technique et science informatiques (TSI)**, Hermes Science Publications.

List of Publications

[1] J. L. Beuchat, J. O. Haenni, H. F. Restrepo, C. Teuscher, F. J. Gómez, and E. Sánchez. Approches matérielles et logicielles de l'algorithme de chiffrement IDEA. *Technique et science informatiques (TSI), Hermes Science Publications*, 21(2):203–224, 2002.

[2] J. L. Beuchat, J. O. Haenni, C. Teuscher, F. J. Gómez, H. F. Restrepo, and E. Sánchez. Une comparaison entre quelques implantations logicielles et matérielles de l'algorithme de chiffrement IDEA. In *Proceedings du 6ème symposium en architectures nouvelles de machines, SympA'6*, pages 25–34, Besancon, France, 19–22 juin 2000.

[3] F. J. Gómez, G. Galeano, H. F. Restrepo, J. O. Haenni, C. Teuscher, and E. Sánchez. Labomat 3: Un entorno completo para el aprendizaje de técnicas de codiseño utilizando una plataforma reconfigurable. In *Actas del Congreso de Tecnologías Aplicadas a la Enseñanza de la Electrónica, TAEE2000*, volume 2, pages 549–552. Universitat Autònoma de Barcelona, Escola Tècnica Superior d'Enginyeria, September 13–15 2000.

[4] J. O. Haenni, C. Teuscher, F. J. Gómez, H. F. Restrepo, and E. Sánchez. Une plate–forme pour l'enseignement et le prototypage d'architectures reconfigurables. In *Proceedings du 5ème symposium en architectures nouvelles de machines, SympA'5*, pages 93–103, Rennes, France, 8–11 juin 1999.

[5] D. Mange, A. Stauffer, G. Tempesti, and C. Teuscher. Dispositif électronique à affichage électro-optique commandé par des circuits logiques programmables. Demande de brevet européen No 01201221.7, March 29, 2001.

[6] D. Mange, A. Stauffer, G. Tempesti, and C. Teuscher. From Embryonics to POEtic machines. In J. Mira and A. Prieto, editors, *Bio-Inspired Applications of Connectionism. Proceedings of the Sixth International Work Conference on Artificial and Natural Neural Networks, IWANN2001*, volume 2085, part II of *Lecture Notes in Computer Science*, pages 1–13. Springer-Verlag, Berlin, Heidelberg, 2001.

[7] B. Mesot and C. Teuscher. Critical values in asynchronous random boolean networks. In W. Banzhaf, T. Christaller, P. Dittrich, J. T. Kim, and J. Ziegler, editors, *Advances in Artificial Life. Proceedings of the 7^{th} European Conference, ECAL2003*, volume 2801 of *Lecture Notes in Artificial Intelligence*, pages 367–376, Berlin, Heidelberg, 2003. Springer-Verlag.

[8] B. Mesot and C. Teuscher. Cellular automata versus random boolean networks. To be submitted to Physica D, 2004.

[9] E. Mosanya, C. Teuscher, H. F. Restrepo, P. Galley, and E. Sánchez. CryptoBooster: A reconfigurable and modular cryprographic coprocessor. In C. K. Koc and C. Paar, editors, *Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems, CHES'99, Worcester, MA*, volume 1717 of *Lecture Notes in Computer Science*, pages 246–256. Springer-Verlag, Berlin, Heidelberg, August 12–13 1999.

[10] B. Petreska and C. Teuscher. A hardware membrane system. In A. Alhazov, C. Martín-Vide, and G. Paun, editors, *Proceedings of the MolCoNet Workshop on Membrane Computing (WMC2003)*, volume 28/03, pages 343–355, Tarragona (Spain), 2003. Rovira i Virgili University, Research Group on Mathematical Linguistics.

[11] B. Petreska and C. Teuscher. A reconfigurable hardware membrane system. In C. Martin-Vide, Gh. Paun, G. Rozenberg, and A. Salomaa, editors, *Workshop on Membrane Computing, WMC2003*, volume 2933 of *Lecture Notes in Computer Science*, Berlin, Heidelberg, 2004. Springer-Verlag. To be published.

[12] H. F. Restrepo, F. J. Gómez, C. Teuscher, J. O. Haenni, and E. Sánchez. Una plataforma reconfigurable para la enseñanza de sistemas lógicos. In *Proceedings of the InterAmerican Conference on Engineering and Technology Education, Intertech2000*, Cincinnati, Ohio, June 14–16 2000.

[13] H. F. Restrepo, R. Hoffmann, A. Pérez-Uribe, C. Teuscher, and E. Sánchez. A networked FPGA-based hardware implementation of a neural network application. In *Proceedings of the IEEE Symposium on Field Programmable Custom Computing Machines, FCCM'00*, pages 337–338, Los Alamitos, CA, April 17–19 2000. IEEE Computer Society.

[14] A. Stauffer, D. Mange, G. Tempesti, and C. Teuscher. BioWatch: A giant electronic bio-inspired watch. In D. Keymeulen, A. Stoica, J. Lohn, and R. S. Zebulum, editors, *Proceedings of the Third NASA/DoD Workshop on Evolvable Hardware, EH-2001*, pages 185–192, Los Alamitos, CA, 2001. IEEE Computer Society.

[15] A. Stauffer, D. Mange, G. Tempesti, and C. Teuscher. A self-repairing and self-healing electronic watch: The BioWatch. In Y. Liu, K. Tanaka, M. Iwata, T. Higuchi, and M. Yasunaga, editors, *Evolvable Systems: From Biology to Hardware. Proceedings of the 4th International Conference on Evolvable Systems, ICES2001, Tokyo, October 3-5, 2001*, volume 2210 of *Lecture Notes in Computer Science*, pages 112–127, Berlin, Heidelberg, 2001. Springer-Verlag.

[16] A. Stauffer, D. Mange, G. Tempesti, and C. Teuscher. Le BioWall. Un tissu informatique pour le prototypage de systèmes bio-inspirés. *Flash Informatique, EPFL*, 4:1–17, 30 avril 2002.

[17] A. Stauffer, D. Mange, G. Tempesti, and C. Teuscher. Sur le BioWall, l'embryonique se décline de façon ludique. *TRACÉS*, 18:26–33, September 18 2002.

[18] A. Stauffer, D. Mange, G. Tempesti, and C. Teuscher. Systèmes informatiques bio-inspirés. Le BioWall: un tissu informatique pour le prototypage de systèmes bio-inspirés. *Bulletin SEV/VSE*, 11:23–27, 2002.

[19] G. Tempesti, D. Mange, A. Stauffer, and C. Teuscher. The Biowall: An electronic tissue for prototyping bio-inspired systems. In A. Stoica, J. Lohn, R. Katz, D. Keymeulen, and R. S. Zebulum, editors, *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware*, pages 221–230, Los Alamitos, CA, 2002. IEEE Computer Society.

[20] G. Tempesti and C. Teuscher. Biology goes digital: An array of 5700 Spartan FPGAs brings the BioWall to "life.". *Xilinx XCell*, 47:40–45, Fall 2003.

[21] C. Teuscher. Study, implementation, and evolution of the artificial neural networks proposed by Alan M. Turing. A revival of his "schoolboy" ideas. Master's thesis, Swiss Federal Institute of Technology Lausanne, Logic Systems Laboratory, EPFL-DI-LSL, CH-1015 Lausanne, February 2000.

[22] C. Teuscher. On the state of the art of POEtic machines. Technical Report 01/375, Swiss Federal Institute of Technology Lausanne, Computer Science Department, CH-1015 Lausanne, November 2001.

[23] C. Teuscher. Künstliches Leben. Migros Genossenschafts Bund, 2002. Kursunterlagen zum "Playground Summer School" Workshop.

[24] C. Teuscher. Turing Day. 90ème anniversaire de la naissance de Alan Mathison Turing. *Flash Informatique, EPFL*, 5:18–19, 4 juin 2002.

[25] C. Teuscher. *Turing's Connectionism. An Investigation of Neural Network Architectures.* Springer-Verlag, London, September 2002.

[26] C. Teuscher, editor. *Alan Turing: Life and Legacy of a Great Thinker.* Springer-Verlag, Berlin, Heidelberg, 2004.

[27] C. Teuscher. Amorphous membrane blending: A new approach to computational blending. In preparation, 2004.

[28] C. Teuscher. *Amorphous Membrane Blending: From Regular to Irregular Cellular Computing Machines.* PhD thesis, Swiss Federal Institute of Technology (EPFL), Lausanne, Switerland, 2004. Thesis No 2925.

[29] C. Teuscher, editor. *BioSystems IPCAT2003 Special Issue.* Elsevier, 2004. To be published in 2004.

[30] C. Teuscher. A new chemical-based hardware architecture. In preparation, 2004.

[31] C. Teuscher. A P-systems implementation on an amoprhous computer. In preparation, 2004.

[32] C. Teuscher. Turing's connectionism. In C. Teuscher, editor, *Alan Turing: Life and Legacy of a Great Thinker*, pages 499–530. Springer-Verlag, Berlin, Heidelberg, 2004.

[33] C. Teuscher and M. S. Capcarrère. On fireflies, cellular systems, and evolware. In A. M. Tyrrell, P. C. Haddow, and J. Torresen, editors, *Evolvable Systems: From Biology to Hardware. Proceedings of the* 5^{th} *International Conference (ICES2003)*, volume 2606 of *Lecture Notes in Computer Science*, pages 1–12, Berlin, Heidelberg, 2003. Springer-Verlag.

[34] C. Teuscher, J. O. Haenni, F. J. Gómez, H. F. Restrepo, and E. Sánchez. A reconfigurable platform for academic purposes. In K. L. Pocek and J. M. Arnold, editors, *Proceedings of the IEEE Symposium on Field Programmable Custom Computing Machines, FCCM'99*, pages 282–283, Los Alamitos, CA, April 21–23 1999. IEEE Computer Society.

[35] C. Teuscher, J. O. Haenni, F. J. Gómez, H. F. Restrepo, and E. Sánchez. A tool for teaching and research on computer architecture and reconfigurable systems. In *Proceedings of the 25th Euromicro Conference,*

volume 1, pages 343–350, Los Alamitos, CA, September 8–10 1999. IEEE Computer Society.

[36] C. Teuscher, D. Mange, A. Stauffer, and G. Tempesti. Bio-inspired computing tissues: Towards machines that evolve, grow, and learn. In *Proceedings of the Fourth International Workshop on Information Processing in Cells and Tissues (IPCAT2001)*, pages 153–164, IMEC, Leuven, Belgium, August 13–17 2001.

[37] C. Teuscher, D. Mange, A. Stauffer, and G. Tempesti. Bio-inspired computing tissues: Towards machines that evolve, grow, and learn. *BioSystems*, 68(2–3):235–244, February–March 2003.

[38] C. Teuscher and E. Sánchez. A revival of Turing's forgotten connectionist ideas: Exploring unorganized machines. In R. M. French and J. P. Sougné, editors, *Connectionist Models of Learning, Development and Evolution. Proceedings of the 6th Neural Computation and Psychology Workshop (NCPW6)*, Perspectives in Neural Computing, pages 153–162, London, 2000. Springer-Verlag.

[39] C. Teuscher and E. Sánchez. Self-organizing topology evolution of Turing neural networks. In G. Dorffner, H. Bischof, and K. Hornik, editors, *Proceedings of the International Conference on Artificial Neural Networks (ICANN2001)*, volume 2130 of *Lecture Notes in Computer Science*, pages 820–826, Berlin, Heidelberg, 2001. Springer-Verlag.

[40] C. Teuscher, E. Sánchez, and M. Sipper. Romero: Un pèlerinage robotique à Santa Fe. In *Proceedings des 11èmes Journées Jeunes Chercheurs en Robotique JJCR11*, pages 145–150. Swiss Federal Institute of Technology Lausanne, 8–9 avril 1999.

[41] C. Teuscher, E. Sánchez, and M. Sipper. Romero's pilgrimage to Santa Fe: A tale of robot evolution. In A. S. Wu, editor, *Workshop Proceedings of the Genetic and Evolutionary Computation Conference, GECCO'99*, pages 409–410, Orlando, Florida, USA, July 13–17 1999.

[42] C. Teuscher, E. Sánchez, and M. Sipper. Romero's odyssey to Santa Fe: From simulation to real life. In M. Jamshidi, A. M. Maciejewski, R. Lumia, and S. Nahavandi, editors, *Robotics and Manufacturing Systems: Recent Results in Research, Development and Applications*, volume 10, pages 262–267, Albuquerque, NM, 2000. World Automation Congres, WAC 2000, TSI Press Series.

[43] C. Teuscher and M. Sipper. Hypercomputation: Hype or computation? *Communications of the ACM*, 45(8):23–24, August 2002.

[44] C. Teuscher and F. Stragiotti. Entwicklung des Windmesscomputers Aiolos II. In M. Wieland, editor, *Schweizer Jugend forscht Jahrbuch*, pages 178–183. Verlag Schweizer Jugend forscht, Winterthur, 1991.

[45] C. Teuscher and U. Teuscher. On Enigmas and oracles: Looking back to the future. *Trends in Cognitive Sciences*, 6(10):410–411, October 2002.

[46] C. Teuscher and U. Teuscher. The Turing Day: Commemorating the anniversary of Alan Mathison Turing's 90^{th} birthday. *European Communications on Mathematical and Theoretical Biology*, 4:27–28, 2002.

[47] U. Teuscher and C. Teuscher. Homme-machine: le jeu du QI perd gagne. *Polyrama*, 117:40–45, December 2002.