Dynamics, Emergent Computation, and Evolution in Cellular Automata

by

Wim Hordijk

Drs., Operations Research, Erasmus University Rotterdam, 1994

DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Computer Science

The University of New Mexico
Albuquerque, New Mexico

December, 1999

©1999, Wim Hordijk

Dedication

Just for fun...

Acknowledgments

First of all, I should apologize to my parents for this dissertation. It is the result of spending close to five years at a 5000 mile (8000 kilometer) distance from them. I know that this was not always easy for them, but their moral and emotional support was nevertheless enormous and invaluable. Thank you so much, for being so understanding and supportive!

This dissertation would never have been possible without the supervision of my two main advisors, Melanie Mitchell and Jim Crutchfield. Over the years, they have supported me in my research activities, have trusted my abilities, showed me alternative directions when I got stuck, and have provided many stimulating discussions and research ideas. They were even willing to correct most of my spelling and grammatical errors ;-) Thanks for providing the opportunity to work with, and learn from, both of you, and for creating the possibility for me to be at the Santa Fe Institute all this time. It truly has made a big dream come true!

I also would like to thank Stephanie Forrest, who was willing to be the chair of the committee during the final stages of my graduate studies. Her objective insights and comments have been very helpful in improving the dissertation. Thanks for taking me under your wing, next to the many graduate students you already had! Also thanks to the other committee members, David Ackley and Andreas Wagner.

They also provided many useful comments for improvements and additions. Thanks for sacrificing some of your valuable time to help me reach this stage.

Most of the research for this dissertation was done at the Santa Fe Institute (SFI). There, I have had numerous opportunities to interact with other students and researchers, and to learn much about other research areas besides computer science. SFI has been a very stimulating and inspiring environment. I especially would like to thank my colleagues in the Evolving Cellular Automata group (next to Melanie and Jim), in particular Jim Hanson, Raja Das, Cris Moore, Erik van Nimwegen, and Cosma Shalizi. Thanks for the stimulating discussions, the camaraderie, and the helpful comments and ideas all through the years. And a special thank you to Jim Hanson for making his cellular automata and computational mechanics code available. Also thanks to everybody else at SFI for making it the special place it is. I will not mention any names, because I'm afraid I will forget someone. You know who you are. Thanks for everything...

At the University of New Mexico I have had the pleasure of interacting with the Adaptive Computation group there. This group of professors, postdocs, and graduate students forms a diverse and stimulating study and research environment. Many seminars and personal discussions within this group have broadened my knowledge of science in areas both within and outside of my own research area. It's been fun interacting with you guys (and girls, of course).

I also want to thank Rémon Sinnema for taking the time and effort to read through the entire first draft of this dissertation. I much appreciate your interest in my work, and your willingness to read and comment on it. It has certainly been helpful!

Doing research and writing a dissertation have been my main goal and occupation over the last several years. However, I could not have done all this without the necessary distractions once in a while. Thanks to Ozric Tentacles for musical support. I have made many trips to "Erpland" and beyond while writing this disser-

tation. Also thanks to the St. John's College Search and Rescue team. It has given me many excuses to get out of my office and into the mountains, looking for lost hikers or evacuating injured backpackers. And thanks to the New Mexico Mountain Bike Adventures for providing another excuse to leave the computer behind once in a while and to jump on my bike to show people the rich cultural and geological history of the Cerrillos Hills. It all has been a lot of fun, and a great experience!

Finally, I would like to give an extra special thanks to Shareen Joshi, for her love and companionship during the last year of my dissertation work. I hope your years of graduate study will be as joyful as mine were. We will see what the future brings...

The research described in this dissertation was supported by the Santa Fe Institute (under ONR grant N00014-95-1-0975) and by grants from the National Science Foundation (IRI-9320200 and IRI-9705853).

Dynamics, Emergent Computation, and Evolution in Cellular Automata

by

Wim Hordijk

Drs., Operations Research, Erasmus University Rotterdam, 1994
Ph.D., Computer Science, University of New Mexico, 1999

Abstract

A new class of models is developed which provides a tool for analyzing emergent computation in cellular automata (CAs). In particular, CAs that were evolved with a genetic algorithm to perform computational tasks requiring global information processing are analyzed. The models are used to make quantitative predictions of the computational performance, on a given task, of these evolved CAs. The models, and the resulting performance predictions, are based on the (emergent) dynamics of the evolved CAs and thus relate the dynamics of these CAs directly to their emergent computational abilities.

Furthermore, the class of models is used to determine quantitatively how and to what extent changes in the emergent dynamics of a CA give rise to changes in its performance. In particular, the differences between the dynamics of CAs that are related to each other in an evolutionary sense are analyzed this way. This, in turn, contributes to a better understanding of the evolution of emergent computation in CAs.

Finally, the class of models itself is investigated more thoroughly. For example, a correctness proof of the models is presented and an expression for scaling the models to larger system sizes is derived. The development, application, and investigation of this class of models thus forms a study of, and provides a better understanding of, the relation among dynamics, emergent computation, and evolution in cellular automata.

Contents

List of Figures **xviii**

List of Tables **xxv**

1 Introduction **1**

1.1 Dynamics, emergent computation, and evolution in decentralized spatially extended systems 3

1.1.1 Decentralized spatially extended systems 3

1.1.2 Dynamics in decentralized spatially extended systems 4

1.1.3 Emergent computation in decentralized spatially extended systems 5

1.1.4 Evolution in decentralized spatially extended systems 7

1.1.5 The relation among dynamics, emergent computation, and evolution in decentralized spatially extended systems 8

1.2 Models of decentralized spatially extended systems and evolutionary processes 8

1.2.1 Cellular automata 9

1.2.2 Genetic algorithms 9

1.3 Dynamics, emergent computation, and evolution in cellular automata 10

1.3.1 Dynamics in cellular automata 10

1.3.2 Computation in cellular automata 11

1.3.3 Dynamics and computation in cellular automata 13
1.3.4 The evolution of emergent computation in cellular automata 14
1.4 A formal study of the relation among dynamics, emergent computation, and evolution in cellular automata 14
1.5 Overview of the dissertation 16

2 Cellular Automata, Formal Languages, and Computational Mechanics 19
2.1 Cellular automata 20
2.1.1 Definitions 22
2.1.2 Cellular automata dynamics 23
2.1.3 The relation between dynamics and computation in cellular automata 25
2.2 Formal languages 26
2.2.1 Alphabets, words, and languages 26
2.2.2 Finite automata 27
2.2.3 Regular languages 29
2.2.4 Finite state transducers 30
2.2.5 Operations on finite automata 31
2.3 Cellular automata as regular language processors 33
2.4 Computational mechanics of cellular automata 37
2.4.1 Regular domains 39
2.4.2 The domain transducer 44
2.4.3 Particles 46
2.4.4 Particle interactions 49
2.4.5 The particle catalog 51

3 Evolving Cellular Automata with Genetic Algorithms 53
3.1 Cellular automata implementation 54

3.2 Computational tasks for cellular automata 55
3.2.1 Density classification 57
3.2.2 Global synchronization–1 58
3.2.3 Global synchronization–2 58
3.2.4 Global synchronization–3 59
3.3 Genetic algorithms 59
3.3.1 The general algorithm 60
3.3.2 Implementation and parameter values 61
3.4 The evolution of emergent computation in cellular automata 66
3.5 The analysis of evolved cellular automata 69
3.6 Results on the new tasks 77
3.7 Related work 81

4 Particle Models of Emergent Computation 85
4.1 The condensation time 87
4.2 Simplifying assumptions for the particle models 89
4.2.1 Particle probability distribution at t_c 89
4.2.2 Zero-width particles 90
4.2.3 Two-particle interactions only 91
4.2.4 Instantaneous interactions 92
4.2.5 Stochastic approximation of phase dependencies 92
4.2.6 Summary 93
4.3 The class of particle models 93
4.4 Predicting the performance of evolved cellular automata 98
4.5 The computational complexity of cellular automata and their particle models 99
4.5.1 The computational complexity of cellular automata 100
4.5.2 The computational complexity of the particle models 103

5 Predicting the Computational Performance of Evolved Cellular Automata **109**

5.1 Performance predictions 110

5.2 Error analysis 116

5.3 Time-to-answer predictions 124

5.4 Comparative analysis 127

5.4.1 Case 1: ϕ_{dens3} and ϕ_{dens4} 128

5.4.2 Case 2: ϕ_{dens4} and ϕ_{dens5} 131

5.4.3 Case 3: ϕ_{sync2} 131

5.4.4 Case 4: ϕ_{parent} and ϕ_{child} 134

5.5 Conclusions 137

6 Further Investigations of the Particle Models **141**

6.1 Correctness of a particle model 143

6.2 Concise approximations of the PPD at t_c 150

6.2.1 A general but inaccurate approximation 152

6.2.2 An accurate approximation for the evolved block expander . 157

6.2.3 An accurate approximation for ϕ_{sync2} 162

6.3 Direct performance calculations 166

6.3.1 The evolved block expander 166

6.3.2 ϕ_{sync2} 168

6.4 Lattice size scaling 169

6.4.1 Condensation time scaling 171

6.4.2 Scaling of the number of particles at t_c 173

6.5 Conclusions 176

7 Conclusions and Discussion **179**

7.1 Summary and conclusions 180

7.2 Discussion 184

7.3 Future work 191

Appendices **193**

A Particle Catalogs **193**

A.1 Density classification 194

A.2 Global synchronization–1 196

A.3 Global synchronization–2 198

A.4 Global synchronization–3 199

B The Update Transducer of $\phi_{\text{bl-exp}}$ **201**

References **205**

List of Figures

2.1 The update process for a cell i in the CA lattice. The update rule ϕ is applied to the cell's local neighborhood configuration η^i to determine the state of cell i at the next time step. 22

2.2 Space-time diagrams of four elementary CAs. Each CA is started with a random initial configuration. 24

2.3 Example of a finite automaton with three states and four allowed transitions. 28

2.4 The CA update transducer T_Φ for elementary CAs. 36

2.5 Space-time diagram of ECA 54 starting from a random initial configuration. 0s are represented as white, 1s as black. 38

2.6 The minimal DFAs corresponding to the regular expressions $(0001)^*$ and $(1110)^*$ representing Λ^{54}. 41

2.7 The minimal DFAs of figure 2.6, representing Λ^{54}, with the temporal transitions added in gray. 42

2.8 The domain transducer for ECA 54, constructed from the DFAs in figure 2.6. Added transitions are shown with thinner arrows than original transitions. 45

2.9 An example of the filtering process using the domain transducer. . . 46

2.10 The filtered version of the space-time diagram of ECA 54 in figure 2.5. 46

2.11 The four particles of ECA 54. (a) The unfiltered appearances. (b) The filtered appearances. Domain symbols are represented by white cells. The inscribed symbols in the black cells refer to the corresponding output symbols of the transitions in the domain transducer. After [HC97]. 48

3.1 Schematic representation of computation in CAs as used here. . . . 55

3.2 The one-point crossover operator. The parts of the bit strings after a randomly chosen crossover point are swapped between the parents a and b, creating two children a' and b'. 65

3.3 The result of a particular GA run on the density classification task. The best fitness versus generation is plotted. Space-time diagrams of CAs that gave rise to significant improvements in fitness during the evolution are shown. After [DMC94]. 68

3.4 The result of a particular GA run on the original global synchronization task. The best fitness versus generation is plotted. Space-time diagrams of CAs that gave rise to significant improvements in fitness during the evolution are shown. After [DCMH95]. 70

3.5 The DFAs representing the regular domains of ϕ_{sync5}. (a) $\Lambda^s = \{0^*, 1^*\}$ and (b) $\Lambda^z = \{(0001)^*, (1110)^*\}$. 71

3.6 The minimal DFAs of figure 3.5, representing Λ^z, with the temporal transitions added in gray. 72

3.7 (a) Space-time diagram of ϕ_{sync5}, starting from a random IC. (b) Same space-time diagram as in (a) with the domains filtered out. After [DCMH95]. 73

3.8 The general strategy of ϕ_{dens5}. (a) White domain smaller than black domain. (b) White domain larger than black domain. 75

3.9 Typical results of evolving CAs for the $\mathbf{T}_{sync2}$ and $\mathbf{T}_{sync3}$ tasks. $\phi_{\text{sync-2a}}$ and $\phi_{\text{sync-3a}}$ are CAs that occurred early on in a GA run on the respective tasks. $\phi_{\text{sync-2b}}$ and $\phi_{\text{sync-3b}}$ are the best CAs found for these tasks. 78

3.10 Space-time diagrams of ϕ_{sync5} and $\phi_{\text{sync-2b}}$, illustrating the similarity in their computational strategies. 80

3.11 An example of a space-time diagram for $\phi_{\text{sync-2b}}$ where the CA never settles down to global synchronization. 81

4.1 Example of the condensation time, marked by the horizontal line labeled t_c. During the condensation phase, there are still some non-domain/particle configurations present in the lattice. 88

4.2 A space-time diagram of ϕ_{sync3} illustrating the simplifying assumptions of the particle models. The labels A, B, C, and D refer to the particle interactions. 91

4.3 A schematic overview of a CA's particle model. 97

5.1 CA and particle model performances for the five CAs from the GA run on the density classification task. 112

5.2 CA and particle model performances for the five CAs from the GA run on the global synchronization–1 task. 114

5.3 CA and particle model performances for the two CAs from the GA run on the global synchronization–2 task, and the two CAs from the GA run on the global synchronization–3 task. 116

5.4 Comparison of a CA space-time diagram (left) with that generated by the particle model (right) for $\phi_{\text{sync-3b}}$. The model accurately simulates the CA's particle dynamics. 118

5.5 Comparison of a CA space-time diagram (left) with that generated by the particle model (right) for ϕ_{dens2}. The circle indicates the area where there is a difference between the CA and its model, which eventually leads to a different overall outcome. 119
5.6 Comparison of a CA space-time diagram (left) with that generated by the particle model (right) for ϕ_{dens5}. The circle indicates the area where there is a difference between the CA and its model, which eventually leads to a different overall outcome. 120
5.7 Comparison of a CA space-time diagram (left) with that generated by the particle model (right) for ϕ_{dens5}. The labels indicate the particles that cause a difference in the overall outcome. 121
5.8 Comparison of a CA space-time diagram (left) with that generated by the particle model (right) for ϕ_{sync5}. The circle indicates the area where there is a difference. 123
5.9 Comparison of a CA space-time diagram (left) with that generated by the particle model (right) for ϕ_{sync5}. The circles indicate the areas where there is a difference. 124
5.10 Example of the difference in interaction time t_i between the CA and its particle model. 126
5.11 The frequency distributions of the total number of particles at t_c for ϕ_{dens3} and ϕ_{dens4}. 130
5.12 Space-time diagrams of ϕ_{parent} (left) and ϕ_{child} (right). The different particle types occurring in the CAs are labeled. 135

6.1 Space-time diagrams illustrating the behavior of $\phi_{\text{bl-exp}}$. 144
6.2 The DFA M_{IC} representing all possible ICs. 146
6.3 The DFA $M_3 = [T^3_{\phi_{\text{bl-exp}}} \circ M_{IC}]_{out}$ representing all possible lattice configurations at time step $t = 3$. 146
6.4 The steps in constructing $M' = [T_{\phi_{\text{bl-exp}}} \circ M]_{out}$. 148

6.5 The sequence of DFAs resulting from applying the FME algorithm iteratively starting with the language $1^*0^61^*$. 149
6.6 Relative frequencies of total number of particles at t_c for ϕ_{dens5} (bars) and as generated by the approximation of the PPD at t_c (solid line). 156
6.7 Space-time diagrams illustrating the behavior of $\phi_{\text{ev-be}}$. 157
6.8 Frequency distributions of the length l of a Λ^1 domain at t_c. Bars indicate the distribution measured over 5,000 ICs with $\rho_0 < 0.5$, and +'s connected by the solid line indicate the distribution measured over 5,000 ICs with $\rho_0 > 0.5$. 160
6.9 A comparison of the relative frequencies of the measured number of particles at t_c (`CA`) and those generated by the PPD model (`Model`). 161
6.10 A comparison of the number of particles at t_c (`CA`) with the numbers generated by the PPD model (`Model`). The top plot shows the results for the number of α particles at t_c, and the bottom plot shows the results for the number of β particles. 165
6.11 Top figure: A schematic (filtered) space-time diagram on lattice size N. Bottom figure: A schematic (filtered) space-time on lattice size $4N$, divided into 4 sublattices of size N. The condensation time of the entire lattice of size $4N$ coincides with the condensation time of the sublattice that condenses last. 171
6.12 Left plot: The empirical probability distribution of t_c on a lattice of size $N = 100$ ($k = 1$). Right plot: The empirical probability distribution of t_c (bars) on a lattice of size $N = 500$ ($k = 5$), and the directly calculated distribution (dots connected by solid line) using order statistics. 173
6.13 Observed values of $\overline{t_c}$ (solid line) and directly calculated values $E[t_c(k)]$ (dashed line) using order statistics. 174

6.14 Observed values for $\overline{n}(k)$ (solid line) and predicted values $E[n(k)]$ (dashed line). The two lines overlap each other exactly. 175

7.1 A space-time diagram of ϕ_{parent}, with its three particles labeled. The bit string representing ϕ_{parent}'s LUT is given below the space-time diagram. Below that, the bits necessary for supporting each of the three particles are given. 185

7.2 A space-time diagram of a CA evolved for the global synchronization–2 task. It has a fitness of 0.83. 187

List of Tables

2.1 The particle catalog of ECA 54. 52

3.1 The GA operators and parameters setting used in the EvCA framework. 66

3.2 The particle catalog of ϕ_{sync5}. 74

4.1 The computational complexities of a CA and two extreme versions of a particle model. The number n of particles at t_c is about two orders of magnitude smaller than the lattice size N. 108

5.1 Performance measurements for the five CAs from the density classification run. The first column gives the CA performance, the second column gives the predicted performance, and the last column shows the percentage difference between the CA and predicted performances. Standard deviations for the performance measurements are given in parentheses. 113

5.2 Performance measurements for the five CAs from the synchronization–1 run. The columns are the same as in table 5.1. 115

5.3 Performance measurements for the four CAs for the synchronization–2 and synchronization–3 tasks. The columns are the same as in table 5.1. 117

5.4 CA and model-predicted $\overline{t_a}$ values, averaged over 10 sets of random ICs. Standard deviations are given in parentheses. 125

5.5 Performance predictions for ϕ_{dens3} and ϕ_{dens4}, interchanging their respective approximations of the PPD at t_c and velocities of the β particle. 129

5.6 Empirically measured probabilities $[\,n]$ of the total number n of particles at t_c for ϕ_{dens2}. 133

5.7 Performance predictions for ϕ_{dens2} as a function of the probability p that a particle at t_c is of type α. $\mathcal{P}_{calc}$ gives the calculated performances using the empirically measured distribution $[\,n]$, and $\mathcal{P}_{model}$ gives the performances as predicted by ϕ_{dens2}'s particle model (standard deviations in parentheses). 134

5.8 CA and model-predicted performances for ϕ_{parent} and ϕ_{child}. 135

6.1 The particle catalog of $\phi_{\text{bl-exp}}$. 145

6.2 The particle catalog of $\phi_{\text{ev-be}}$. 158

6.3 Empirically measured probability distribution $[\,n]$ of the total number n of Λ^1 domains at t_c. 159

6.4 The empirically measured probability distribution $[\,n]$ of the total number n of particles at t_c. 164

6.5 The empirically measured conditional probabilities $[\,n_\alpha|n]$ of the number n_α of type α particles given a total number n of particles at t_c. . 164

Chapter 1

Introduction

Many systems in nature produce complicated patterns, which emerge from the local interactions of relatively simple individual components that live in some spatially extended world. Notably, this type of emergent pattern formation often occurs without the existence of a central control. Such systems, consisting of (many) components in a spatially extended world, with local interactions only and no central control, are generally referred to as decentralized spatially extended systems. Examples of emergent pattern formation in such systems include the foraging and nest-building behavior of social insects [DG89, Bon98], spiral waves in cultures of amoebae [Dev89b, Win90], synchronized oscillations in the brain [Gra94, LD94], etc.

Emergent pattern formation in decentralized spatially extended systems often entails an important functionality for the system as a whole. In other words, the emergent patterns give rise to some form of globally coordinated behavior, or global information processing, which is used by the system to sustain itself or make certain decisions. For example, by means of emergent patterns, an ant colony decides what the shortest path is to some food source, amoebae decide when and where to aggregate to reproduce or find the highest concentrations of food, and the brain classifies sensory inputs (these examples are elaborated below).

This global information processing in decentralized spatially extended systems, mediated by emergent pattern formation, is known as *emergent computation* [For90b]. These pattern forming behaviors, and the resulting emergent computations, have evolved over time. In other words, many decentralized spatially extended systems have adapted, under the force of natural selection, to use their tendency to produce patterns to perform certain kinds of global information processing that benefit the system as a whole. However, there is little understanding of how the dynamics (i.e., the spatio-temporal behavior) of decentralized spatially extended systems gives rise to emergent computation, or even how such systems and their behaviors are produced by an evolutionary process.

This dissertation investigates these relations among dynamics, emergent com-

putation, and evolution in decentralized spatially extended systems. This investigation is done in the context of using genetic algorithms to evolve cellular automata. Cellular automata (CAs) are one of the simplest models of decentralized spatially extended systems in which emergent patterns are observed, and in which emergent computation can take place (see e.g. [FTW84, Gut91, Wol94]). Genetic algorithms (GAs) [Hol75, Gol89, Mit96] are a simple model of an evolutionary process, and can be used to evolve CAs to perform certain tasks that require global information processing. A new class of models is then developed and used to analyze the relation between dynamics and emergent computation in GA-evolved CAs. These models are used to make quantitative predictions about the evolved CAs' computational performances, based on the CAs' emergent dynamics. These modeling results provide a better understanding of how emergent patterns can give rise to global information processing, and how evolution can produce decentralized spatially extended systems that use their pattern forming behavior to perform emergent computation.

1.1 Dynamics, emergent computation, and evolution in decentralized spatially extended systems

1.1.1 Decentralized spatially extended systems

A *spatially extended system* is a system made up of a (often large) number of individual components that exist in some spatial, n-dimensional world. For example, ants in an ant colony live in a nest that exists in a three-dimensional world.

Because of the spatial extent of these worlds, direct communication between individual components is often limited to local interactions. For example, an ant in a three-dimensional nest can communicate only with other ants that happen to be in its own local neighborhood.

A *decentralized system* is one in which no single component or group of components controls the entire system. Instead, the actions of each individual component depend solely on its own internal behavior and its direct interactions with the other components with which it can communicate. For example, an artificial neural net (a system for performing computations) is a decentralized system. Such a system consists of a number of individual units (neurons), generally with local connections between these units. Each neuron is in a certain state which changes over time depending on the states of its neighbors (i.e., other neurons with which it is connected). There is no single controlling neuron in the system, and information processing is carried out in a distributed and parallel fashion (for an introduction to computing with neural nets, see e.g. [Lip87]). In contrast, a standard von Neumann style computer is a centralized system, since there is one central processing unit (CPU) that controls the behavior and actions of all other units in the computer.

So, combining these definitions, a *decentralized spatially extended system* is a system made up of a (large) number of individual components in which communication between components is limited to local interactions, and in which there is no central controlling component or group of components. Examples of such systems include many chemical reaction systems, physical systems like spin glasses, and biological systems like the brain, bacterial colonies, ant colonies, etc.

1.1.2 Dynamics in decentralized spatially extended systems

Many decentralized spatially extended systems, both physical and biological, have a tendency to generate complicated patterns [Win90, CPM95, NNS97]. Well known examples of pattern formation in physical systems include the spiral waves in the chemical Belousov-Zhabotinski reaction [Win87], and the Raleigh-Bénard convection cells in fluid dynamics [Man90]. Examples in biological systems include self-organizing patterns in the foraging behavior and nest building of social insects [DG89, Bon98], the spiral waves that appear during the spontaneous aggregation of a reproductive

multicellular organism from individual amoebae in the life cycle of Dictyostelium discoideum [Dev89b, Win90], and the synchronized oscillations of neural assemblies in the brain [Gra94, LD94].

The global patterns in these systems arise out of the local actions and interactions of the relatively simple (as compared to the system as a whole) individual components, without the existence of a central control. Often, these local interactions are nonlinear. In other words, the behavior of the system as a whole is more than simply a linear superposition of the behaviors of the individual components when considered in isolation. This property is captured by the popular phrase "the whole is more than the sum of the parts". The global patterns appear only when the system as a whole is active, through the local actions and interactions of the individual components.

Thus, this global *dynamics* (i.e., the emergent pattern formation) is neither explicitly specified in the equations of motion of the individual components (the local "rules" by which they act) nor in the system's boundary conditions. In this way, the global, pattern forming, dynamics in decentralized spatially extended systems can be considered an *emergent* property of these systems [Cru94b, O'C94].

1.1.3 Emergent computation in decentralized spatially extended systems

Often, the emergent patterns in decentralized spatially extended systems give rise to some form of globally coordinated behavior, or global information processing. In the foraging behavior of ants, for example, the colony "decides" what the shortest path is from the nest to a particular food source. This decision is based on self-organizing patterns in the ants' foraging behavior; patterns which, in turn, are mediated by individual ants depositing pheromone, a chemical signal to other ants [DG89]. Nest building in termites is accomplished by a similar process [Bon98]. In the spontaneous aggregation of individual amoebae to form a reproductive slime mold, spiral waves

emerge through local cell-cell interactions by means of chemical signaling. Based on these patterns, the amoebae implicitly "decide" when and where to aggregate and how to differentiate into two types of cells: those that make up the stalk, and those that make up the spores in the fruiting body [Dev89b, Win90]. Synchronized oscillations of neural assemblies are thought to play an important role in encoding information [Gra94]. For example, one locust species "classifies" different types of odors by coherent oscillations in different ensembles of neurons [LD94].

These "decisions" and "classifications" can be considered emergent computations. In [For90b], *emergent computation* is defined as consisting of:

1. A collection of agents, each following explicit instructions;
2. Interactions among the agents (according to the instructions), which form implicit global patterns at the macroscopic level, i.e., epiphenomena;
3. A natural interpretation of the epiphenomena as computations.

The above examples clearly conform to these three constituents: (1) There is a collection of agents (ants, amoeba, neurons) that act according to specific instructions or rules; (2) There are interactions between the agents that give rise to implicit global patterns (foraging paths, spiral waves, synchronized oscillations); and (3) These patterns are used to store, transmit, and process information to make decisions or classifications at a global level, which can be naturally interpreted as computations. Thus, many decentralized spatially extended systems can be considered to perform emergent computation mediated by emergent pattern formation.

It is claimed that systems that exhibit emergent computation have several potential advantages over traditional computing systems in terms of efficiency, flexibility, and robustness [MRH86, For90b]. Increased efficiency is possible, since these systems are massively parallel. The different parts of the system all act (compute) in parallel and different parts of the computation can occur simultaneously in different parts of the system. Increased flexibility is possible, since the individual components

in the underlying system are relatively simple and interact only locally, which can make it easier to modify or add parts to the system without having to worry directly about global constraints or dependencies. And robustness can be increased, since the system as a whole can often still function correctly when one or a few individual components in the system fail, whereas a traditional computing system typically breaks down entirely when one of the subsystems fails. These advantages seem to be exploited to a large degree in natural decentralized spatially extended systems such as the ones mentioned above. In artificial systems, however, these advantages are usually hard to achieve in practice. The implicit nature of the emergent computation makes such systems hard to "program". See [For90a] for an overview of both natural and artificial systems that exhibit emergent computation.

1.1.4 Evolution in decentralized spatially extended systems

The decentralized spatially extended systems that we see in nature today, in particular biological ones, have evolved. Shaped by the process of natural selection, the individuals in these systems have adapted over time to take advantage of their collective emergent pattern forming behavior to perform global information processing that benefits the system as a whole. The reason evolution has produced so many decentralized spatially extended systems capable of emergent computation, as opposed to systems that perform global information processing in a centralized, fully connected way, is likely because of the mentioned advantages of efficiency, flexibility, and robustness for emergent computation as opposed to centralized computation. Alternatively, it could be that it is simply too difficult for evolution to produce fully centralized systems from scratch.

1.1.5 The relation among dynamics, emergent computation, and evolution in decentralized spatially extended systems

There is little theoretical understanding of how the dynamics (i.e., the behavior over time) of a decentralized spatially extended system gives rise to emergent computation. In other words, it is unclear how emergent patterns in a system's dynamics encode, transmit, and process the necessary information to perform computations, or at least how we can interpret these patterns in such a way. Thus, an important question is: *What is the relation between dynamics (in particular, the emergent pattern formation) and computational ability in decentralized spatially extended systems?*

Furthermore, it is not well understood how evolution has produced decentralized spatially extended systems capable of emergent computation. In other words, it is not clear why the emergent patterns formed by one system are better adapted to performing certain tasks than the emergent patterns generated by another system. Therefore, another important, and also largely unresolved question is: *How does evolution take advantage of a system's inherent dynamics to produce emergent computation in decentralized spatially extended systems?*

These two questions about the relations among dynamics, emergent computation, and evolution in decentralized spatially extended systems form the main motivation for the work presented in this dissertation.

1.2 Models of decentralized spatially extended systems and evolutionary processes

To allow a formal study of the relation among dynamics, emergent computation, and evolution in decentralized spatially extended systems, mathematical and computational models are used here. In particular, cellular automata are used as a class of

models of decentralized spatially extended systems, and genetic algorithms are used as a class of models of evolutionary processes. These models can be viewed as idealized versions of decentralized spatially extended systems and evolutionary processes, respectively, and are arguably the most simple systems that still exhibit the properties and behaviors that are of interest here.

1.2.1 Cellular automata

A cellular automaton (CA) is a discrete dynamical system consisting of a regular lattice of "cells" in some dimension d. Each cell in the lattice can be in one of a finite number of states. At discrete time steps, all cells update their states simultaneously, based on a local update rule which is the same for all cells. This update rule takes as input the current local neighborhood configuration of a cell (i.e., the states of a cell and its nearest neighbors), and returns the state the cell will be in at the next time step. This process of simultaneously updating the cells in the lattice is repeated over time, starting from some particular (random) initial configuration of cell states. When plotted over time, the lattice as a whole can show a wide variety of behaviors, depending on the particular local update rule that is used.

CAs are used as models of the behavior of a wide variety of decentralized spatially extended systems, including fluid dynamics, pattern formation in chemical and biological systems, pattern recognition, traffic flow, voting behavior, and emergent behavior in so called complex systems in general (see e.g. [FTW84, Gut91, Wol94] for overviews). Furthermore, CAs are easily implemented on a computer and, since they are well defined, lend themselves directly to mathematical analysis. Section 2.1 reviews CAs in more detail.

1.2.2 Genetic algorithms

Genetic algorithms (GAs) are stochastic search methods inspired by biological evolution [Hol75, Gol89, Mit96]. GAs maintain a population of candidate solutions, often

represented as bit strings. Each individual in the population is assigned a fitness value that reflects how well it solves a given problem. Based on these fitness values, certain individuals from the current population are selected and used to create new individuals (offspring) by applying genetic operators such as crossover and mutation. The idea is to "evolve" solutions by repeated application of the selection and reproduction operators.

GAs have been used extensively to find satisfactory solutions to many (primarily numerical, multi-parameter) optimization problems. But they have also been used successfully to model certain aspects of natural evolution (see, e.g., the various GA conference proceedings for an overview [Gre85, Gre87, Sch89, BB91, For93, Esh95, Bäc97]). Like CAs, GAs are easily implemented on a computer and lend themselves to some mathematical analysis. Section 3.3 reviews GAs in more detail, in particular in the context of evolving CAs.

1.3 Dynamics, emergent computation, and evolution in cellular automata

1.3.1 Dynamics in cellular automata

Although simply defined, CAs can exhibit a wide range of behaviors, from fixed point or simple periodic behavior to highly complex or even "chaotic". Based on this classification scheme from dynamical systems theory, Wolfram proposed a qualitative classification of CA behavior into four classes, intending to capture all possible CA behavior [Wol84b]:

1. A (fixed) homogeneous state.
2. A set of separated simple stable or periodic structures.
3. A "chaotic" behavior.

4. Complex localized structures, sometimes long-lived.

Subsequently, Langton tried to make this classification more quantitative by introducing the λ parameter, a statistic of the CA update rule. He studied the relation between the "average" dynamics of CAs and their λ values. Langton speculated that Class 4 behavior occurs at "critical" λ values, which he associated with a "phase transition between periodic and chaotic behavior" [Lan90]. However, the association between this phase transition and critical λ values, as reported in [Lan90], does not appear to be very strong, since there is a rather wide range of λ values where these transitions between the two kinds of behavior occur. Indeed, Wolfram's classification scheme and the usefulness of Langton's λ parameter have been questioned by others [HC92, CH93, MHC93].

1.3.2 Computation in cellular automata

CAs can be considered a computational system and their computational capabilities can be investigated just as for Turing machines, random access machines, Post production systems, etc. The computational aspects of CAs have indeed been studied in great depth (see, e.g., [Wol84a, Nor89, CIHY90a]).

Different kinds of computation can be distinguished in CAs. First, the input-output behavior over time of a CA can be interpreted as a computation. In this case, the CA update rule itself is viewed as the "program", while the input to the computation is encoded in the initial configuration, which is transformed by the CA into some "desired" output in the form of a spatial configuration at some later time step. Examples of this kind of computation include language recognition [Smi72, Ter94] and "soliton-like" computation [SKW88, SS94].

Second, it has been shown that certain CAs, given certain special initial configurations, are capable of universal computation. For example, the well known two-dimensional CA called the "Game of Life" [Gar83] was proved to be a universal computer by using special structures like "gliders" and "glider guns" to implement

logical gates [BCG82]. Furthermore, it has been proved for several one-dimensional CAs that they are computationally universal by simulating some other computationally universal device inside the CA, for example a Turing machine [LN90] or some kind of production system [Coo00].

Third, there is a notion of *intrinsic computation* in CAs. This does not necessarily involve a "useful" computation, but refers to the detection of structural components, embedded in the dynamics of a CA, that encode, transmit, and process information. These components, or patterns, can be described in computational terms using formal language and automata theory. In this *computational mechanics* framework, the dynamics of a CA is thus analyzed and described in terms of notions from computation theory, without the need to attach any semantics or usefulness to the discovered structures [HC92, CH93, Han93, HC97]. The computational mechanics approach and this notion of intrinsic computation is reviewed in more detail in section 2.4.

Chapter 4 in [GBG+98] provides an elaborate review of all these different kinds of computation in CAs. These computations may or may not be emergent. For example, the soliton-like computation in CAs is explicitly constructed by using different CA cell-states to encode the absence or presence of, and the interactions between, different kinds of soliton-like particles with which the computation is performed. Thus, this computation would not be considered emergent (in the definition of [For90b] used above). In the "Game of Life" emergent structures called gliders and glider guns are used to create explicitly a particular initial configuration such that the CA dynamics mimics computation with logical gates. This seems to be somewhere in between emergent and explicitly programmed. The intrinsic computation embedded in a CA's dynamics, however, appears to be truly emergent.

1.3.3 Dynamics and computation in cellular automata

Because CAs have the capacity for a wide range of dynamics as well as many kinds of computation, they form an ideal class of models to study the relation between dynamics and computational ability in decentralized spatially extended systems. Wolfram's classification of CA behavior and Langton's λ parameter, together with their respective speculations about computationally capable CAs, provided a first step in this direction. However, both these studies dealt with "generic" or "average" CA behavior only and, moreover, did not use a well-defined classification scheme or measure of computational capability. Therefore, ultimately they did not provide a direct or quantitative relation between a CA's dynamics and its computational ability.

Packard addressed some of these issues by using a GA to evolve CAs to perform a specific computational task [Pac88]. His results appeared to support Langton's hypothesis that CAs capable of useful computation occur near critical λ values. However, even though a direct measure of computational capability was now available (i.e., the ability of the CAs to solve the given task), Packard did not relate the dynamics of the evolved CAs to this computational capability. He looked only at the distribution of λ values in the final generation of a GA run.

Trying to replicate Packard's results, Mitchell and colleagues found results that contradict those of Packard [MHC93]. Using theoretical arguments and empirical evidence, they showed that useful computation in CAs does not necessarily occur near critical λ values. They then continued with an investigation of (1) the evolutionary history of the evolved CAs, and (2) the relation between the actual dynamics of the evolved CAs and their computational capability, or performance, on the given task [MHC93, MCH94a, MCH94b]. These studies constitute the first concrete steps towards a better understanding of the relation among dynamics, computation, and evolution in CAs.

1.3.4 The evolution of emergent computation in cellular automata

In the first series of experiments by Mitchell *et al.*, the CAs evolved by the GA showed no form of sophisticated *emergent* computation. In [MCH94b] some of the impediments that prevented the GA from finding such CAs are discussed. However, in subsequent experiments CAs with sophisticated emergent strategies for performing the given task were found by the GA [DMC94, DCMH95, CM95]. Both the emergent strategies in these CAs and the evolutionary history that gave rise to these strategies were analyzed in more detail in [Das98, CMD98].

The emergent strategies of these evolved CAs were described in terms of *regular domains*, *particles*, and *particle interactions*[1]. These are emergent structures embedded in the CA's dynamics, by which, it was claimed, the computational task and the necessary global information processing are performed. Furthermore, the evolutionary history of these evolved CAs was explained in terms of changes in the occurrences, velocities, and interactions of these domains and particles. This provided the next step towards a better understanding of the relation among dynamics, emergent computation, and evolution in CAs. This work on evolving CAs is reviewed in more detail in chapter 3.

1.4 A formal study of the relation among dynamics, emergent computation, and evolution in cellular automata

The work on evolving CAs with GAs and the subsequent analyses have provided much insight into the relation among dynamics, emergent computation, and evolution in

[1] The notions of regular domains and particles are formalized in the computational mechanics for CAs framework, which is reviewed in more detail in section 2.4.

CAs. However, so far the explanations and claims have not been verified directly by, for example, any quantitative predictions of a CA's computational performance based on its pattern-forming behavior. To give an example, the following claim was made when explaining the computational strategies of some of the CAs that appeared during the GA evolution: "In the succeeding generations, the velocity of the particles and their interactions play the most crucial role in determining the fitness of a CA rule" [DMC94]. So far, this claim has not been verified in any quantitative way. In [CMD98], some of the mechanisms of emergent computation in evolved CAs (in terms of particles and their interactions) have been identified and indicated in more detail, supporting the above claim. But these results are specific to the particular CAs that were investigated, and did not yet provide a general framework for investigating the relation between dynamics and emergent computation in evolved CAs in a more quantitative and predictive way. Thus, what is still needed is a general framework for analyzing this relation more formally and mathematically, so that:

1. The concept of an emergent strategy in a CA is well-defined and can be formalized in an algorithmic procedure;

2. This algorithmic procedure can be used to make quantitative predictions of an evolved CA's computational performance;

3. This algorithmic procedure can be used to relate changes in the emergent strategy of a CA to changes in the CA's computational performance in a quantitative way;

4. These changes can be related to the evolutionary history of the evolved CAs, i.e., the algorithmic procedure can be used to explain why and to what extent one emergent strategy is better than another.

The research described in this dissertation addresses these important aspects, and thus goes beyond previous work on evolving cellular automata. It involves the

design, implementation, and application of a class of *particle models* for analyzing the emergent strategies of CAs that are evolved by a GA to perform certain computational tasks. This class of models is based on the computational mechanics framework and uses the regular domains and particles, as observed in the evolved CAs' dynamics, to construct an algorithm for simulating an evolved CA's emergent strategy. This algorithm is then used to make predictions of the evolved CAs' computational performance and to relate changes in a CA's emergent strategy directly to corresponding changes in performance. Furthermore, the algorithm is used to perform comparative analyses of evolutionarily related CAs, indicating to what extent differences in the emergent strategies of these CAs contribute to differences in their performances. The development and subsequent use of the particle models thus provides a means to study formally the relation among dynamics, emergent computation, and evolution in cellular automata.

1.5 Overview of the dissertation

In the next chapter, CAs are reviewed in detail. Also, a brief overview of formal languages and finite automata is provided, which is necessary for the subsequent review of the computational mechanics framework. This framework provides an analysis tool for detecting and classifying emergent structures in the dynamics of CAs. Chapter 3 then provides a detailed overview of the evolving cellular automata framework, reviews previous results, and presents new results on evolving CAs on additional computational tasks. Other work related to evolving CAs is also reviewed briefly. The class of particle models for analyzing the emergent computational strategies of evolved CAs is then fully introduced in chapter 4. This chapter also includes a comparison between CAs and their corresponding particle models in terms of their computational complexity. Chapter 5 presents results on using the particle models to analyze the relation between dynamics and emergent computation in evolved CAs quantitatively.

In particular, the particle models are used to predict the computational performances of evolved CAs, and to analyze the differences in the emergent strategies of evolutionarily related CAs. Chapter 6 presents a further investigation of the actual class of particle models itself. This includes, among other issues, a proof in the most basic setting of the correctness of a particle model, and a derivation of an expression to predict the scaling of certain quantities with the CA lattice size. Finally, chapter 7 summarizes the main conclusions, presents some discussion, and suggests future work.

Chapter 2

Cellular Automata, Formal Languages, and Computational Mechanics

In this chapter, cellular automata, formal languages, and computational mechanics are reviewed in more detail. Cellular automata are discrete-state dynamical systems which form a general class of models of decentralized spatially extended systems. Formal languages are a concept from theoretical computer science. The class of formal languages that is most useful here are regular languages, which are sets of words that can be represented by the mathematical framework of finite automata. Cellular automata can be viewed as regular language processors. In other words, their global dynamics can, in principle, be studied using finite automata. However, as is shown below, this approach is cumbersome and impractical for most situations of interest. Computational mechanics offers an alternative way of formally studying the dynamics of CAs by identifying and classifying regularities that appear in a CA's global behavior. These regularities, expressed in terms of regular languages, form a concise description of the CA's global, or emergent, behavior.

2.1 Cellular automata

Cellular automata (CAs) are dynamical systems that are discrete in state, space, and time. Originally, CAs were introduced by von Neumann, after a suggestion by Ulam, to study the logical organization behind biological self-reproduction [vN66]. After von Neumann's death, Burks completed and extended this original work on CAs [Bur70]. This early work on CAs was mainly theoretical.

CAs were popularized in a series of Scientific American articles about the "Game of Life", a two-dimensional CA invented by Conway [Gar83]. With the fast increase in processor speed and availability of cheap memory that followed the development of the first computers, interest in CAs also increased. These technological improvements made it easier and faster to run CAs on a computer, and the complicated patterns they generate could now be observed directly. With this development, CA research shifted to modeling natural and artificial systems and building simula-

tions (see e.g. [MBVB90]). The introduction of special CA hardware even made very fast simulation of large-scale systems possible [TM87].

CAs, because of their pattern generating properties, are an interesting class of systems in and of themselves. However, they can also be regarded as an alternative to differential equations for modeling physics [Tof84], and as a model of parallel distributed computation [Hil84]. CAs have been used in many different ways, including

- studying the logical organization behind self-reproduction [vN66, Lan84],
- the basis of new computer architectures [Hil84, TM87],
- connections with formal languages [Nor89, CIHY90a],
- traffic simulations [SN98],
- models of voting behavior [BM96],
- studying fluid dynamics [MTV86],
- studying pattern formation [TH88, BH91],
- modeling and simulation of a wide variety of physical systems [Vic84, MBVB90],
- modeling and simulation of a wide variety of biological systems [EEK93],

and many, many other areas. For a more extensive survey of CA applications, see chapter 2 in [CCNC97].

In this dissertation, CAs are used as models of decentralized spatially extended systems. They have the capability of generating a wide range of dynamics and they are capable of performing many kinds of computation. For this reason, they form an ideal tool for studying the relation between dynamics and computation in decentralized spatially extended systems. Although a CA can be defined in any arbitrary but finite spatial dimension, only one-dimensional CAs are considered here. The mathematical framework reviewed below is therefore restricted to one-dimensional CAs.

2.1.1 Definitions

A *one-dimensional cellular automaton* consists of a linear lattice, or array, of identical *cells*, each of which can be in one of a finite number k of states. The (local) *state* of cell i at time t is denoted $s_t^i \in \Sigma = \{0, 1, \ldots, k-1\}$. The (global) state $\mathbf{s}_t$ of the CA at time t is the *configuration* of the entire spatial lattice, $\mathbf{s}_t = (s_t^0, s_t^1, \ldots, s_t^{N-1}) \in \Sigma^N$, where N is the (possibly infinite) size of the lattice.

At each time step, all the cells in the lattice update their state simultaneously according to a *local update rule* ϕ. This update rule takes as input the *local neighborhood configuration* η of a cell, which consists of the states of the cell i itself and its $2r$ nearest neighbors (r cells on either side). So, at site i the local neighborhood is $\eta^i = (s^{i-r}, \ldots, s^i, \ldots, s^{i+r})$. r is called the *radius* of the CA. The update rule ϕ then returns the new state of cell i, $s_{t+1}^i = \phi(\eta_t^i)$. Figure 2.1 gives a graphical illustration of this update process.

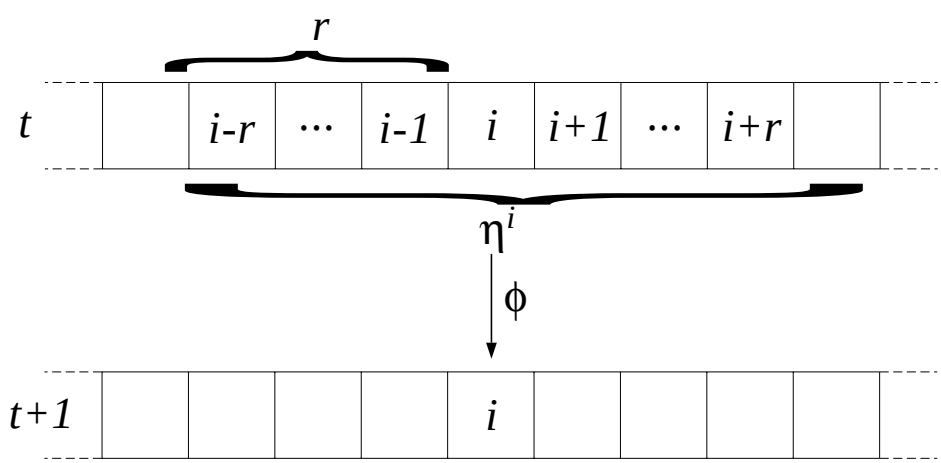

Figure 2.1: The update process for a cell i in the CA lattice. The update rule ϕ is applied to the cell's local neighborhood configuration η^i to determine the state of cell i at the next time step.

The local update rule ϕ, which is the same for each cell in the lattice, can be represented as a *lookup table* (LUT) which lists all the possible local neighborhood configurations ($|\eta| = k^{2r+1}$), together with their respective output states s_{t+1}. The *global update rule* $\Phi : \Sigma^N \to \Sigma^N$ applies ϕ in parallel to all cells in the CA lattice,

$\mathbf{s}_{t+1} = \Phi(\mathbf{s}_t)$. For finite N it is also necessary to specify boundary conditions. Here, periodic boundary conditions are used, i.e., $s^{N+i} = s^i$.

The simplest class of CAs are the *elementary* CAs (ECAs), for which $(k, r) = (2, 1)$. Such a CA has $2^{2r+1} = 2^3 = 8$ entries in its lookup table. Since the output state of each of these entries can be either a 0 or a 1, there are $2^8 = 256$ ECAs. These ECAs have been studied in great detail.

In [Wol83], Wolfram introduced a convenient numbering scheme for elementary CAs. First, all possible neighborhood configurations η are written as binary numbers and listed in decreasing order (see the example below). For each possible value for η, the corresponding output state (a 0 or a 1) is written. This list of 0s and 1s is then also considered a binary number, and its corresponding decimal number is the *rule number* of the CA. For example, for ECA 18 this rule number construction looks like:

η_t	111	110	101	100	011	010	001	000	
s_{t+1}	0	0	0	1	0	0	1	0	$= 2^4 + 2^1 = 16 + 2 = 18$

Of course, this scheme works both ways: from the lookup table one can find the rule number and from the rule number one can easily construct the lookup table.

2.1.2 Cellular automata dynamics

The dynamics of a (one-dimensional) CA can be visualized in a *space-time diagram*, where the global states $\mathbf{s}_t$ of the CA are plotted over time, with time increasing down the page. CAs can show a wide variety of behaviors, from fixed point or simple periodic behavior to highly complex or even "chaotic". Figure 2.2 presents space-time diagrams of four different elementary CAs (ECA 160, 184, 18, and 105), showing this variety in behavior, even in this simplest class of CAs. In each space-time diagram, the CA is started with a randomly generated initial configuration (IC).

Wolfram proposed a qualitative classification of (asymptotic) CA behavior into four classes, intending to capture all possible CA behavior [Wol84b]:

1. A (fixed) homogeneous state.

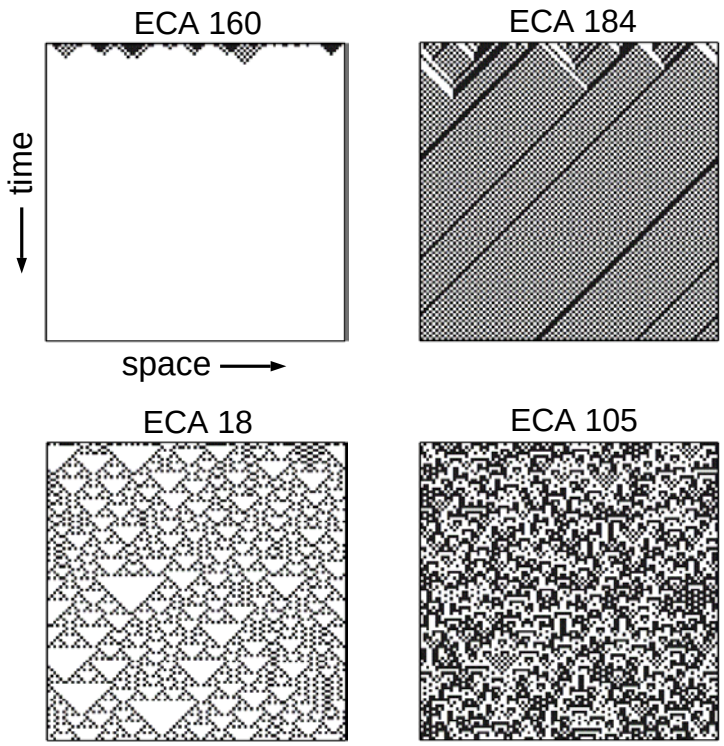

Figure 2.2: Space-time diagrams of four elementary CAs. Each CA is started with a random initial configuration.

2. A set of separated simple stable or periodic structures.

3. A "chaotic" behavior.

4. Complex localized structures, sometimes long-lived.

For example, ECA 160 in figure 2.2 is in class 1. ECA 184 is an example of a class 2 CA. Both ECA 18 and 105 are classified as chaotic, or class 3 CAs. An example of a class 4 CA is the two-dimensional "Game of Life" CA mentioned in the previous chapter. These four classes are analogs of corresponding classes of behavior from dynamical systems theory (see e.g. [Dev89a]). Wolfram's classes are phenomenological, though, and in his scheme CAs are classified only by visual inspection of space-time diagrams.

Langton tried to make Wolfram's classification more quantitative by introducing the λ parameter [Lan86]. This parameter is a statistic of the output states in the

CA lookup table, defined as the fraction of *non-quiescent* states in this table. The *quiescent* state of a CA is an arbitrarily chosen state $s \in \Sigma$. Concretely, for two-state CAs (i.e., $\Sigma = \{0, 1\}$), if state 0 is chosen as the quiescent state, the λ value is simply the fraction of 1s in the output states in the lookup table. Another interpretation of λ is that if an (infinite length) CA is started with a random initial configuration, such that all possible neighborhoods η occur with equal probability, then λ is the fraction of 1s in the CA lattice at the next time step.

Analogous to the occurrence of a bifurcation sequence in a dynamical system when some control parameter is increased, Langton observed that when CA lookup tables are created at random, but with increasing λ values, the "generic" dynamics of these CAs go through Wolfram's classes in the following order: $1 \rightarrow 2 \rightarrow 4 \rightarrow 3$. Langton argued that there is an obvious mapping between the λ parameter and Wolfram's classes, where low to intermediate λ values are associated with classes 1 and 2, intermediate or "critical" λ values with class 4, and high λ values with class 3. The "critical" λ values, Langton claimed, are associated with a phase transition between periodic and chaotic CA behavior [Lan86, Lan90, Lan91].

2.1.3 The relation between dynamics and computation in cellular automata

As mentioned earlier, CAs are capable of many kinds of computation. Both Wolfram and Langton speculated that useful, or even universal, computation would be possible only in class-4 CAs, that is, CAs at "critical" λ values. However, both these studies dealt only with "generic" or "average" CA behavior that is classified by visual inspection. Moreover, they did not use a well-defined measure of computational capability. Langton used a correlation measure based on mutual information between cells in the lattice, which in itself is a well-defined and useful measure. However, he then states that "if cells are cooperatively engaged in the support of a computation, they must exhibit some–but not *too* much–correlation in their behaviors" (original

italics) [Lan90]. Unfortunately, Langton did not quantify what "some–but not *too* much–correlation" is.

So, the association between Wolfram's four classes and Langton's λ parameter is based only on average behavior judged by visual inspection, and thus does not appear to be very strong, and certainly not quantitative. Subsequently, this classification scheme and the usefulness of the λ parameter were questioned by others [HC92, CH93, MHC93]. Because these studies did not use a well-defined or quantitative measure of computational ability in CAs, they left the question remaining of how to formalize the relation between a CA's dynamics and its computational ability.

2.2 Formal languages

This section reviews the topic of formal languages, in particular regular languages, which are important concepts from theoretical computer science. This background is necessary for an understanding of the last part of this chapter on computational mechanics, and also for some material in later chapters. Most of the review in this section can be found in more detail in any standard textbook on formal language and automata theory (e.g., [HU79, CL89, Mor98]). This section serves mainly as a reminder and to fix notation.

2.2.1 Alphabets, words, and languages

An *alphabet* Σ is a finite set of symbols. Examples include the 26 letters of the English alphabet, $\Sigma = \{a, b, c, \ldots, z\}$, and the binary alphabet, $\Sigma = \{0, 1\}$. A *word* w over the alphabet Σ is a finite sequence of symbols from Σ. For example, *cat* and *house* are words over the English alphabet, and so is $pfrrt$, even though it might not have a meaning to us. Similarly, 0000 and 1010101 are words over the binary alphabet. The ith symbol of word w is denoted w_i. The *length* of a word w, denoted $|w|$, is defined as the number of symbols in the word. For example, $|cat| = 3$ and $|1010101| = 7$.

There is one special word of length zero, the *empty word*, denoted ε, which exists for every possible alphabet. The set of all possible words over an alphabet Σ is denoted Σ^* (meaning an arbitrary number of symbols from Σ).

A *formal language* L over the alphabet Σ is a (possibly infinite) subset of Σ^*, $L \subseteq \Sigma^*$. In other words, a formal language is a set of words over some alphabet. For example, all of the following sets are languages over the binary alphabet: $\emptyset$ (the empty language), $\{\varepsilon\}$ (the language consisting only of the empty word), $\{\varepsilon, 0, 00, 000, 0000, \ldots\} = 0^*$, $\{0, 1, 00, 01, 10, 11\}$, etc.

2.2.2 Finite automata

Formally, a *finite automaton* (FA) is defined as a 5-tuple:

$$M = \{Q, \Sigma, \delta, q_0, F\}$$

where Q is a finite set of *states*, Σ is an alphabet, $q_0 \in Q$ is the *initial* state, $F \subseteq Q$ is a set of *final* or *accepting* states, and $\delta : Q \times \Sigma \to Q$ is a *transition function*, written $\delta(q, a) = q'$, which takes a state $q \in Q$ and a symbol $a \in \Sigma$ to another state $q' \in Q$. A finite automaton can be graphically represented by circles (for the states) connected by arrows (for the transitions). Figure 2.3 shows an example with 3 states (denoted by the circles labeled a,b, and c, i.e., $Q = \{a, b, c\}$), alphabet $\Sigma = \{0, 1\}$, start state a (denoted by the unlabeled incoming arrow), and final state c ($F = \{c\}$, denoted by the double circle). The *allowed* transitions are shown by arrows going from one state to another, labeled with symbols. In this example $\delta(a, 0) = a$, $\delta(a, 1) = b$, $\delta(b, 1) = c$, and $\delta(c, 0) = a$ are the allowed transitions. The other transitions that are possible in principle (e.g. $\delta(b, 0) = c$), but that are not explicitly specified, are the *disallowed* transitions.

A finite automaton can be used in two ways: in an *input* (or *reading*) mode or in an *output* (or *writing*) mode. In the input mode, symbols from a word w are read one by one. The automaton starts in its initial state q_0 and reads the first

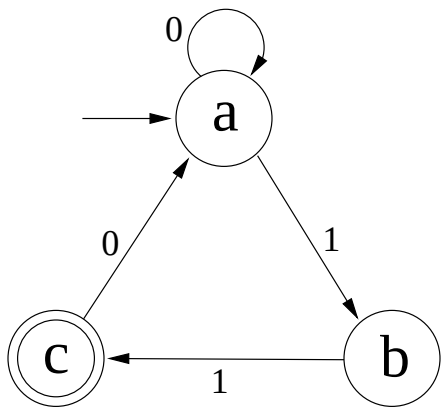

Figure 2.3: Example of a finite automaton with three states and four allowed transitions.

symbol, w_1. It then makes a transition to another state q' according to the transition function: $q' = \delta(q_0, w_1)$. The automaton then reads the next symbol, w_2, and makes a transition to state $q'' = \delta(q', w_2)$, and so on for every next symbol w_i in the word w. This way, the automaton reads, or *scans*, the entire word w, until no symbols are left. The automaton is said to *accept* a word w if, after reading the entire word, it ends in an accepting state $q \in F$. If it is not in an accepting state after reading the entire word, or if a disallowed transition is encountered while reading the word, the automaton is said to *reject* the word. For example, the FA in figure 2.3 will accept the word 0110011, but reject the word 0111011. In the input mode, a finite automaton can be used as a *recognizer* of words.

In the output mode, the automaton again starts in the initial state q_0, but will write a word w instead of reading one. Initially the word w is the empty word ε. At each step, the automaton makes a transition out of the current state q' to one of the states that can be reached from q' via an allowed transition. The destination state q'' is chosen in some way, e.g., at random, out of all the states reachable from q'. Next, the symbol w_i that labels the chosen transition is added to the end of the word w. This process can terminate only when the automaton is in a final state. In this mode, a finite automaton can be used as a *generator* of words.

One final notion, important in the following, is the distinction between a *deter-*

ministic finite automaton (DFA) and a *nondeterministic* finite automaton (NFA). In a DFA, there is at most one outgoing transition from each state labeled with a symbol a for each $a \in \Sigma$. In an NFA, there can be more than one outgoing transition from one particular state labeled with the same symbol. So, every DFA is also an NFA, but not every NFA is a DFA (although an NFA can be converted into an equivalent DFA, as shown below). The example in figure 2.3 shows a DFA.

2.2.3 Regular languages

A *regular language* L is a formal language for which there exists some finite automaton $M(L)$ that accepts all words in L and rejects all words not in L. Note that a language of finite size is always regular. For any regular language L there exists a unique deterministic finite automaton $M_{min}(L)$ with a minimal number $|Q|$ of states that accepts L. Generally, the subscript is omitted and it is assumed that $M(L)$ is the minimal automaton that accepts L. In a similar way, for every finite automaton M there is a corresponding regular language $L(M)$ that consists of all (and only) the words that are accepted by M.

The regular language $L(M)$ associated with the automaton M of figure 2.3 consists of all the words where 1s occur only in pairs, with one or more 0s in between the pairs of 1s, and where all words end with a pair of 1s.

Another convenient way to represent regular languages are regular expressions. A *regular expression* $R(L)$ represents a regular language L using primitives and operations. The primitives are ε (the empty word), the symbols a (for all $a \in \Sigma$), and the parentheses (). The operations are + (for union, or the `OR` operation), $\cdot$ (dot; for concatenation), and * (for zero or more repetitions). Often the $\cdot$ for concatenation is omitted and $a \cdot b$ is just written as ab. Also, sometimes the notation $^+$ is used for one or more repetitions, i.e., $a^+ = aa^*$. Regular expressions are built by using these primitives and operations. As with FAs, for every regular expression R there is a corresponding regular language $L(R)$ that consists of all the words that can be built

using R.

It turns out that finite automata and regular expressions are equivalent. In other words, for every regular expression $R(L)$ there exists a finite automaton $M(L)$ that accepts L (and only L), and vice versa. The regular expression equivalent to the finite automaton of figure 2.3 is $0^*(110^+)^*11$. Sometimes it is more convenient to represent regular languages with finite automata, and sometimes with regular expressions. Since they are equivalent, however, they can be used interchangeably.

A particular subclass of the regular languages are process languages. A *process language* is a regular language L where, in the corresponding minimal DFA $M(L)$, all states are both initial and accepting states. This minimal DFA $M(L)$, representing a process language, is called a *process graph*.

2.2.4 Finite state transducers

A finite automaton can either be used in input mode or in output mode. Naturally, it is also possible to combine these two modes into one automaton, such that for each transition a symbol is read as well as written. The resulting automaton is referred to as a *finite state transducer* (FST). Formally, an FST is defined as a 7-tuple:

$$T = \{Q, \Sigma_{in}, \Sigma_{out}, \delta, \lambda, q_0, F\}$$

where Q, δ, q_0, and F are as in the definition of a finite automaton, Σ_{in} is the input alphabet, Σ_{out} is the output alphabet, and $\lambda : Q \times \Sigma_{in} \rightarrow \Sigma_{out}$ is the *observation function*, written $\lambda(q, a) = b$, which maps a state $q \in Q$ and an input symbol $a \in \Sigma_{in}$ to an output symbol $b \in \Sigma_{out}$.

The representation of a FST is similar to that of a finite automaton, except that there are now two symbols associated with each transition: an input symbol and an output symbol. The arrows representing the allowed transitions have labels $a|b$, where $a \in \Sigma_{in}$ denotes the input symbol and $b \in \Sigma_{out}$ denotes the output symbol.

A FST effectively implements a mapping f_T from one language (a set of words

over Σ_{in}) to another language (a set of words over Σ_{out}). Given a word w over the input alphabet Σ_{in}, the FST scans this word and for each symbol w_i it reads it makes the appropriate state transition and outputs a symbol w'_i. It thus maps a word $w \in \Sigma_{in}$ to a word $w' \in \Sigma_{out}$.

2.2.5 Operations on finite automata

NFA-to-DFA conversion

As mentioned earlier, a finite automaton can be deterministic (DFA) or nondeterministic (NFA), and every DFA is also an NFA but not necessarily vice versa. However, DFAs and NFAs are provably computationally equivalent. In other words, for every NFA M there is an equivalent DFA M'. Equivalent again means that $L(M)$ and $L(M')$ are the same.

There is an algorithm for converting any given NFA into an equivalent DFA. Briefly, the procedure is to have the states Q' in the DFA M' correspond to *subsets* of the set of all states Q in the NFA M, i.e., $Q' \subseteq 2^Q$, where 2^Q is the *power set*[1] of Q. Specifically, the state $q' \in Q'$ that the DFA M' is in after reading part of a word w is the state corresponding to the subset of all states $q \in Q$ that the NFA M can be in after reading the same part of w. From this algorithm it follows that converting an NFA into its equivalent DFA, in principle, can cause an exponential increase in the number of states (from $|Q|$ to $2^{|Q|}$).

Minimization of DFA

When constructing a DFA $M(L)$ that accepts a given regular language L, it is not guaranteed that $M(L)$ is minimal. Similarly, when converting an NFA M to an equivalent DFA M', it is not guaranteed that the resulting automaton M' is minimal.

[1] The power set 2^Q of a set Q is the set of all possible subsets of Q. For example, if $Q = \{1, 2, 3\}$, then $2^Q = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$.

However, for every DFA M there exists a unique minimal DFA M_{min} that is equivalent to M, i.e., $L(M_{min}) = L(M)$.

Similarly to the NFA-to-DFA conversion, there exists an algorithm to find the minimal DFA M_{min} for any given DFA M. Briefly, this algorithm consists of finding pairs of *equivalent* states. Two states are equivalent, for example, when the sets of all (sub)words that can lead to these two states are identical. These equivalent states can then be merged into one state, thus reducing the number of states in M. This process is done iteratively. Furthermore, given an arbitrary DFA M, there could be states in M that can never be reached. These states and their outgoing transitions can be deleted, again reducing the number of states in M.

FST output language

Given a finite state transducer T, what is the set of all words that T can produce (i.e., what is T's *output language*), given all possible inputs? The *output automaton* $[T]_{out}$ is defined as the minimal DFA that accepts the set of all words that can be "emitted" from T on all possible inputs. $[T]_{out}$ can be constructed as follows [Han93]:

1. The input symbols on all transitions of T are disregarded, leaving only the output symbols labeling transitions.

2. The resulting automaton is typically an NFA. This NFA is converted to a DFA, using the NFA-to-DFA conversion algorithm.

3. The resulting DFA is then minimized using the DFA minimization algorithm.

This construction is called *minimization with respect to the output.*

Automata composition

Given a finite automaton $M(L)$ and an FST T, $M(L)$ can be used in output mode to generate words that can then be read by T. This results in restricting the domain

of the mapping f_T to the language L. The automaton that represents this restricted mapping is the composition $T \circ M(L)$. This composition operator is formally defined as follows [Han93]:

Let M and T be a finite automaton and a finite state transducer, respectively:

$$M = \{Q^M, \Sigma^M, \delta^M, q_0^M, F^M\} \text{ and}$$

$$T = \{Q^T, \Sigma_{in}^T, \Sigma_{out}^T, \delta^T, \lambda^T, q_0^T, F^T\}, \text{ with } \Sigma_{in}^T = \Sigma^M.$$

The *composition* $T' = T \circ M$ of T and M is a transducer

$$T' = \{Q', \Sigma_{in}', \Sigma_{out}', \delta', \lambda', q_0', F'\}$$

such that: Q' are ordered pairs of the states in Q^T and Q^M, i.e., $Q' = Q^T \times Q^M$; the input alphabet is $\Sigma_{in}' = \Sigma_{in}^T = \Sigma^M$; the start state is $q_0' = (q_0^T, q_0^M)$; the set of final states is $F' = \{(q^T, q^M) : q^T \in F^T, q^M \in F^M\}$; the output alphabet is $\Sigma_{out}' = \Sigma_{out}^T$; the transition function $\delta' : Q' \times \Sigma_{in}' \to Q'$ is given by

$$\delta'(q', \sigma_{in}) = (\delta^T(q^T, \sigma_{in}), \delta^M(q^M, \sigma_{in})), \text{ where } q' = (q^T, q^M)$$

and the observation function $\lambda' : Q' \times \Sigma_{in}' \to \Sigma_{out}'$ is given by

$$\lambda'(q', \sigma_{in}) = \lambda^T(q^T, \sigma_{in}), \text{ where } q' = (q^T, q^M).$$

The minimization with respect to the output construction can then be used to find the minimal DFA that represents the language that can be produced from T', i.e., $[T']_{out}$.

2.3 Cellular automata as regular language processors

There is a direct connection between CAs and regular languages (see e.g. [Nor89, CIHY90a, CIHY90b]). In particular, CAs can be viewed as *regular language processors*. In this view, the lattice configurations $\mathbf{s}_t$ are considered words from some regular language that are mapped onto each other by the CA dynamic.

First, recall that the global CA update rule Φ operates on individual lattice configurations $\mathbf{s}_t$. In other words, the global CA update rule Φ maps one particular lattice configuration $\mathbf{s}_t$ to another lattice configuration $\mathbf{s}_{t+1}$ at the next time step by applying the local update rule ϕ simultaneously to each cell s_t in the lattice. So, starting with a specific initial configuration $\mathbf{s}_0$, the CA goes through a sequence of lattice configurations $\{\mathbf{s}_0, \mathbf{s}_1, \mathbf{s}_2, \dots\}$ which is completely determined by $\mathbf{s}_0$ and Φ, with $\mathbf{s}_t = \Phi\mathbf{s}_{t-1} = \Phi^t\mathbf{s}_0$.

Alternatively, an *ensemble operator* $\mathbf{\Phi}$ can be defined that operates on *sets* of lattice configurations $\Omega_t = \{\mathbf{s}_t\}$ [Wol84a]. In other words, the ensemble operator $\mathbf{\Phi}$ maps one set of lattice configurations Ω_t to another set of lattice configurations Ω_{t+1} at the next time step by applying the global update rule Φ to each lattice configuration $\mathbf{s}_t$ in Ω_t:

$$\Omega_{t+1} = \{\mathbf{s}_{t+1} : \mathbf{s}_{t+1} = \Phi(\mathbf{s}_t), \forall \mathbf{s}_t \in \Omega_t\}$$

For example, if Ω_0 is some set of lattice configurations, then $\Omega_1 = \mathbf{\Phi}\Omega_0$ is the set of lattice configurations that can occur at time step $t = 1$ given that the CA starts with an initial configuration $\mathbf{s}_0$ from Ω_0. So, starting with ICs from a specific set Ω_0, the CA lattice configurations that can be observed at each time step t form the sequence $\{\Omega_0, \Omega_1, \Omega_2, \dots\}$ which is completely determined by Ω_0 and $\mathbf{\Phi}$, with $\Omega_t = \mathbf{\Phi}\Omega_{t-1} = \mathbf{\Phi}^t\Omega_0$.

Considering lattice configurations $\mathbf{s}_t$ as words over the CA alphabet Σ, a set of lattice configurations Ω_t can now be considered a formal language. In particular, when the lattice size N is finite these sets are *regular* languages, since there are only a finite number of lattice configurations possible on a finite lattice. For example, the set of all $|\Sigma|^N$ possible initial configurations on a lattice of size N forms the regular language $\Omega_0 = \Sigma^N$. It can be shown that the finite iterates $\mathbf{\Phi}^t\Omega_0, t < \infty$, of any regular language Ω_0 are also regular languages [Wol84a, HC92]. Thus, a (finite-size) CA can be viewed as a regular language processor.

So, the ensemble operator $\mathbf{\Phi}$ forms a mapping from one regular language Ω_t to

another regular language Ω_{t+1} and can thus be represented as a finite state transducer T_Φ. This FST, referred to as the *CA update transducer*, can, for example, be used to read the current lattice configuration $\mathbf{s}_t$ of a CA Φ, and write the lattice configuration $\mathbf{s}_{t+1}$ to which $\mathbf{s}_t$ is mapped by Φ. In fact, this method provides the most (space) efficient way of updating a CA lattice. Alternatively, T_Φ can be used to construct $M(\Omega_{t+1})$ explicitly, given $M(\Omega_t)$, using the automata composition and FST output language operators introduced in the previous section. This construction process is formalized in the *finite machine evolution (FME) algorithm*[2] [Han93]: Given the CA update transducer T_Φ for a specific CA rule Φ and a finite automaton $M_t = M(\Omega_t)$, the finite automaton $M_{t+1} = M(\Omega_{t+1}) = M(\mathbf{\Phi}\Omega_t)$ is constructed as

$$M_{t+1} = [T_\Phi \circ M_t]_{out}$$

Figure 2.4 shows the CA update transducer T_Φ for elementary CAs. Since elementary CAs have a radius $r = 1$, three $(2r + 1)$ consecutive symbols $\eta_t^i = \{s_t^{i-1}, s_t^i, s_t^{i+1}\}$ need to be read before the updated symbol $s_{t+1}^i = \phi(\eta_t^i)$ can be written. So, the last two symbols read need to be remembered while scanning a CA lattice, which requires four states in the FST (recall that the CA alphabet is binary for elementary CAs). Thus, the update transducer for an ECA consists of four states. The (allowed) transitions between these states are labeled with an input symbol in $\Sigma = \{0, 1\}$ and an output symbol $\phi(s^{i-1}, s^i, s^{i+1})$, which, of course, depends on the actual ECA rule used.

For example, suppose T_Φ, while scanning a CA lattice, is currently in state `00`. This means that the last two symbols read were both 0. Now suppose that the next symbol read is a 1. The update transducer T_Φ then follows the outgoing transition (from state `00`) labeled with input symbol 1. This transition leads to state `01`, since the last two symbols read are now 01. Furthermore, this transition has output symbol $\phi(001)$, since the local neighborhood $\eta = 001$ was just read. Depending on the actual

[2] A finite automaton is sometimes also called a finite state machine, or just finite machine. These names all mean the same.

ECA rule ϕ used, this output symbol is either 0 or 1. For ECA 18, for example, this symbol would be a 1, since $\phi_{18}(001) = 1$.

The update transducer for general (k, r) CAs is similarly constructed, but using k^{2r} states to remember the last $2r$ symbols read. Also note that the FST shown in figure 2.4 is for infinite-lattice CAs. For finite lattices additional transducer states are required to take the CA's boundary conditions into account.

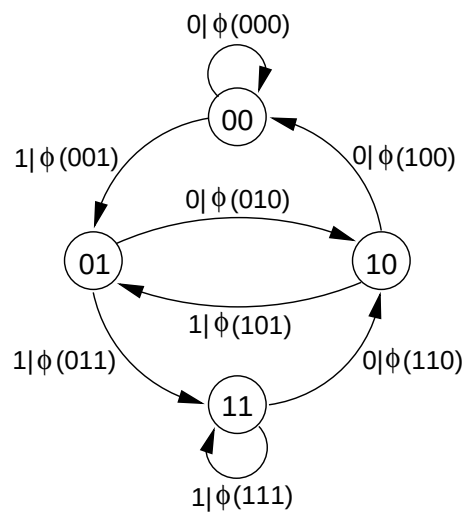

Figure 2.4: The CA update transducer T_Φ for elementary CAs.

As mentioned, the update transducer T_Φ of a CA can be used to construct $M(\Omega_{t+1})$ explicitly given $M(\Omega_t)$. So, in principle, the FME algorithm can be used, starting with $\Omega_0 = \Sigma^*$, to study a CA's global behavior Ω_t over time, without the need to actually iterate the CA update rule on all possible ICs. This would be analogous to the study of the evolution of state ensembles in dynamical systems theory. In practice, however, the sizes (i.e., number of states $|Q|$) of the finite automata $M_t = M(\Omega_t)$ often explode. For example, Wolfram calculated the number of states in $M_t = M(\Omega_t)$ for t up to five, using $\Omega_0 = \Sigma^*$, for the elementary CAs [Wol84a]. For some ECAs this number, after only five iterations, is estimated to be larger than 20,000!

So, characterizing the global behavior of a CA by studying its state space using $\mathbf{\Phi}$ (or equivalently T_Φ) acting on $\Omega_0 = \Sigma^*$ is implausible for anything but the

simplest CAs. Instead, one could try to apply $\mathbf{\Phi}$ to other, more restricted Ω_0 to, for example, identify the regularities that characterize a CA's global dynamics. In particular, this method can be used to identify the dynamically-homogeneous space-time patterns that dominate a CA's behavior [HC92]. This is exactly what is done in the computational mechanics approach for analyzing the dynamics of CAs.

2.4 Computational mechanics of cellular automata

In many CAs, it turns out, there are only a few important patterns, i.e., regularities in the space-time behavior, that dominate the global behavior of the system as a whole. For example, in ECA 160, shown in figure 2.2, the CA dynamics quickly settles down to a stable configuration of all 0s. In ECA 184, shown in the same figure, the dominating pattern is that of a "checkerboard" (alternating 0s and 1s). In ECA 18, same figure, the behavior is dominated by white triangles of all sizes. In more complicated CA behaviors, the dynamics is often dominated by a number of different patterns that can occur simultaneously.

These dominating patterns form a *pattern basis*, i.e., a set of regularities in terms of which the global dynamics of the CA can be described. This is analogous to using notions of attractors, basins, and separatrices to describe a system's behavior in dynamical systems theory. *Computational mechanics* (CM) for cellular automata provides a formal framework for discovering, expressing, and classifying a CA's pattern basis by combining tools from dynamical systems theory and computation theory. Most of these tools have already been introduced in the previous sections, in particular the ensemble operator $\mathbf{\Phi}$ and the finite machine evolution algorithm.

Computational mechanics is a general framework for analyzing dynamical systems that allows one to infer a model of the hidden process that generated the observed behavior of a system. This model captures the patterns, or regularities, observed in this behavior. In the CM for CAs framework, the dominating patterns in a CA's

dynamics are identified and then classified into *regular domains*, *particles*, and *particle interactions*. These different classes of patterns are explained in detail below. Suffice to say here that these patterns, once identified, are expressed in terms of regular languages which can then be represented by DFAs. This representation reveals how (and how much) information is stored in the system, and how this information is transformed over time. It is this aspect that makes computational mechanics "computational", in the sense of "computation theoretic". The "mechanics" in computational mechanics comes from its conceptual ties to statistical mechanics in physics (for details, see [SC99]).

For a complete introduction to the CM for CAs framework, see [HC92, CH93, Han93, HC97]. In this section, the CM framework is reviewed and illustrated with an example using ECA 54. This CA was analyzed earlier in [BNR91]. In [HC97], ECA 54 was analyzed again, but using the CM framework. The example in this section largely follows that of [HC97]. Figure 2.5 shows a space-time diagram of ECA 54, starting from a random initial configuration.

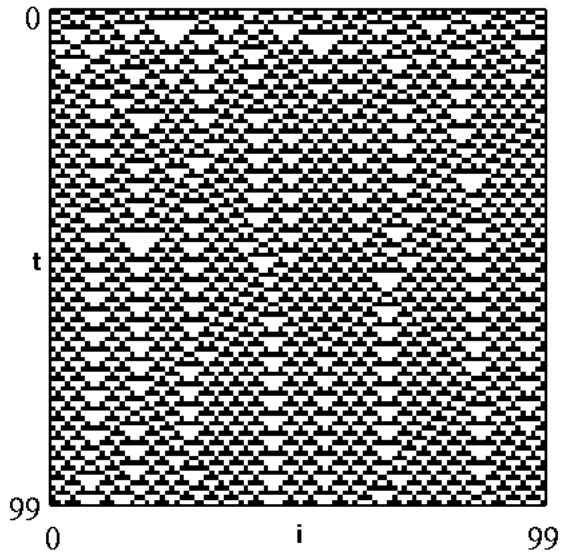

Figure 2.5: Space-time diagram of ECA 54 starting from a random initial configuration. 0s are represented as white, 1s as black.

2.4.1 Regular domains

The first, and perhaps most important, class of patterns in the CM framework used to describe the dynamics of a CA is that of *regular domains*. Informally, a regular domain is a homogeneous region of space-time in which the same set of spatial configurations appear over and over again, both in time and in space. Examples of such homogeneous regions are the all-0s configuration in ECA 160 (the spatial configuration '0' repeated) and the checkerboard regions in ECA 184 (the spatial configuration '01' repeated). This notion of regularity, or pattern, is formalized in the definition of a regular domain as follows.

A regular domain Λ of a CA Φ is a process language[3] representing a set of spatial configurations, with the following two properties:

1. *Temporal invariance or periodicity*: Λ is mapped onto itself by the CA dynamic, i.e., $\mathbf{\Phi}^p\Lambda = \Lambda$ for some finite p.

2. *Spatial homogeneity*: The process graph of each temporal period of Λ is strongly connected (i.e., there is a path between every pair of states).

The meaning of the first property, temporal invariance, is that a regular domain is a periodic, i.e., temporally repeating, set of spatial configurations represented by a regular language (possibly with period one, in which case the domain is invariant). Note that this periodicity is a "language" periodicity. It indicates which spatial configurations that make up the regular domain are mapped into which other ones. However, it does not indicate exactly which symbol of one spatial configuration is mapped into which symbol of another spatial configuration. In other words, it does not indicate the relative positions of these spatial configurations in the CA lattice over time. This issue is addressed below in the example.

The second property of a regular domain, spatial homogeneity, is a form of spatial translation invariance. In other words, the regular domain, or pattern, is

[3]Recall from section 2.2.3 that a process language is a regular language in which in the corresponding DFA, the process graph, all states are both start and final states.

position independent and can occur anywhere in the lattice. Furthermore, a domain (or rather, an instance of a domain) can be extended indefinitely (assuming an infinite lattice). Finally, inside a regular domain there is no notion of absolute position, only of relative position. Compare this to walking on a railroad track with identical cross beams. All you can tell is whether you are on a cross beam or in between two beams. There is no way of telling on *which* beam you are in an absolute sense. There is only a notion of relative position (on or between beams). But what is known, is that when you are on a beam, the next position will be in between two beams, followed by being on a beam again, etc.

Since a regular domain Λ is a regular language (in particular, a process language), it can be represented by either a set of regular expressions or by their corresponding minimal DFAs. Of course, a CA can have more than one regular domain. The set of all regular domains of a particular CA is denoted $\mathbf{\Lambda}$.

For many CAs, candidates for a regular domain can be identified by visual inspection of the CA's space-time diagram. In more complex cases, an automated inference technique, called ϵ-machine reconstruction [CY89, Cru94a], can be used to discover candidate domains in the space-time behavior of a CA. Once such a candidate domain Λ^c has been identified for a given CA Φ, either by eye or by using the ϵ-machine reconstruction method, it then has to be shown that this candidate domain has the two properties of a regular domain. The first property (temporal periodicity) can be verified using the FME algorithm to show that Λ^c is closed (i.e., mapped onto itself) under the CA dynamic Φ. The second property (spatial homogeneity) is verified by constructing the process graph representing Λ^c, and checking whether it is strongly connected. If the candidate domain Λ^c indeed has these two properties, then it is classified as a regular domain Λ of Φ.

Note the difference between a set of lattice configurations Ω_t and a regular domain Λ. The elements of Ω_t represent *entire* lattice configurations that can occur at time step t. The elements of Λ, on the other hand, represent (local) spatial

configurations that occur repeatedly *within* the CA lattice at any time step. Both sets, however, are represented by regular languages, and their respective DFAs can be composed with a CA update transducer to study their dynamics under a certain CA update rule.

As an example, consider the space-time diagram of ECA 54 in figure 2.5. Close inspection of this space-time diagram shows many alternating and repeating spatial configurations of three-white-one-black and three-black-one-white cells. This pattern can be represented by the set of regular expressions $\Lambda^{54} = \{(0001)^*, (1110)^*\}$. Since this is a recurring pattern in the CA's behavior, it forms a good candidate for being a regular domain of ECA 54.

To verify the first property (temporal periodicity), it is easy to show, using the update transducer $T_{\Phi_{54}}$ of ECA 54 and the FME algorithm, that $(0001)^*$ is mapped into $(1110)^*$ and vice versa [HC97]. Figure 2.6 shows the minimal DFAs corresponding to the regular expressions representing Λ^{54}. Clearly, both DFAs are strongly connected (every state is reachable from every other state), and the second property (spatial homogeneity) is also verified. So, Λ^{54} can indeed be classified as a regular domain of ECA 54.

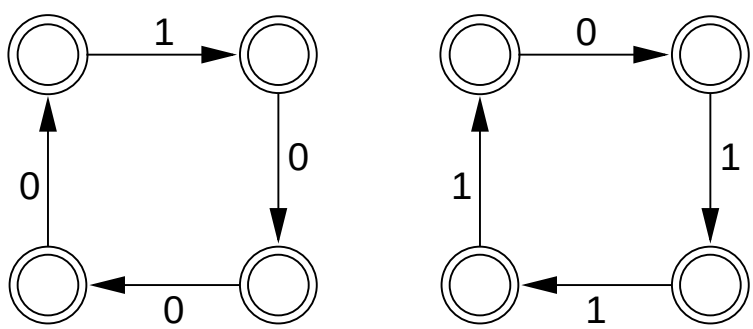

Figure 2.6: The minimal DFAs corresponding to the regular expressions $(0001)^*$ and $(1110)^*$ representing Λ^{54}.

Note that the regular expression representation of a regular domain Λ is somewhat ambiguous. For example, in the above example for ECA 54, Λ^{54} is represented with the set of regular expressions $\{(0001)^*, (1110)^*\}$. However, this could just as well

have been written as $\{(1000)^*, (0111)^*\}$, or $\{(0100)^*, (1101)^*\}$, or any of the possible permutations. It can be shown that $(0001)^*$ is mapped into $(1110)^*$, and vice versa, by $T_{\Phi_{54}}$, but what is lacking in the regular expression representation is an indication of *how* they get mapped into each other.

This lacuna can be filled by using the DFA representation of a regular domain. For example, in the DFAs of figure 2.6, next to the already existing spatial transitions, temporal transitions can be added that indicate how the two spatial configurations are mapped into each other by the CA dynamic. The result of adding these temporal transitions is shown in figure 2.7. Note that the two DFAs are drawn in different relative positions, compared to figure 2.6, to facilitate adding the temporal transitions, which are shown in gray.

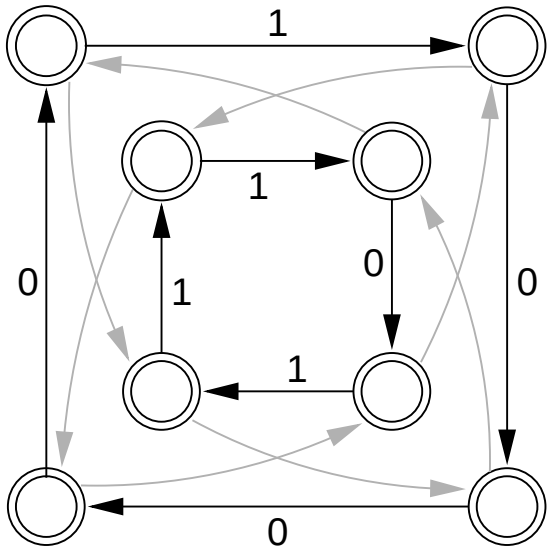

Figure 2.7: The minimal DFAs of figure 2.6, representing Λ^{54}, with the temporal transitions added in gray.

Each regular domain Λ has a temporal and a spatial periodicity. The *temporal* periodicity of a domain Λ is equal to the number of regular expressions it contains. More formally, the temporal periodicity p^t_Λ is the smallest value p for which $\mathbf{\Phi}^p\Lambda = \Lambda$. Under the CA dynamic Φ, a domain Λ continually cycles through its p^t_Λ *phases* in a fixed order. These phases can be (arbitrarily) numbered $1, \ldots, p^t_\Lambda$. There is a

corresponding regular expression (and minimal DFA) for each one of these phases.

The *spatial* periodicity p^s_Λ of a domain Λ is the number of cells in the CA lattice after which each temporal phase of the domain repeats itself. Since each cell in the lattice has the same local update rule, the spatial periodicities of all the temporal phases of a domain Λ are equal. Thus, the minimal DFAs representing a domain's temporal phases all have a number of states that is equal to this spatial periodicity p^s_Λ.

Since Λ^{54} consists of two alternating patterns, $(0001)^*$ and $(1110)^*$, that are mapped into each other, its temporal periodicity is $p^t_{\Lambda^{54}} = 2$. Furthermore, since there are four states in each of the minimal DFAs in figure 2.6, its spatial periodicity is $p^s_{\Lambda^{54}} = 4$.

Recall that the domain Λ^{54} for ECA 54 was first inferred by visual inspection, and then proved to be a regular domain. So, one obvious question is: Are there any other regular domains for ECA 54? In fact, there are. For example $\Lambda = \{0^*\}$ is also a regular domain of ECA 54. Since $\phi_{54}(000) = 0$, it is clear that a configuration of 0^* is mapped onto itself by ECA 54. Furthermore, since the minimal DFA for 0^* has only one state, it is trivially strongly connected. So, $\Lambda = \{0^*\}$ is a regular domain with both temporal and spatial periodicities equal to one. However, it is an unstable domain. It occurs only on very specific initial configurations, and even if it does occur in a space-time diagram, it will quickly disappear and be "taken over" by other patterns such as the other domain Λ^{54}. Likewise, $\Lambda = \{(0011)^*\}$ is another unstable domain of ECA 54 [HC97].

So, it is clear that a CA can have multiple regular domains, some of which are stable and occur often in an actual space-time diagram, and some of which are unstable and occur only for a small set of very specific ICs. It is therefore possible that, when constructing the set of all domains $\mathbf{\Lambda}$, some domains are missed simply because they were not observed in the CA's space-time diagrams. For the purpose of modeling the dynamics of evolved CAs and predicting their computational performance with the

particle models introduced in chapter 4, such rare domains do not form an important contribution since these performance predictions are averaged over a large number of random ICs. As it turns out, not including these rare domains in the description of a CA's global behavior does not have significant consequences for the purposes of the current study. It remains an open question, however, whether it is possible to construct an algorithm, for example, that will find all regular domains given a particular CA update rule ϕ.

2.4.2 The domain transducer

Given one or more regular domains Λ^i for a given CA Φ, a *domain transducer* can be built using the DFAs $M(\Lambda^i),\ i = 1, \ldots, |\mathbf{\Lambda}|$. This transducer can then be used to map raw space-time configurations to "filtered" configurations where the domain regularities are suppressed. The resulting filtered space-time diagram reveals where these domain regularities are violated.

Briefly, the construction of the domain transducer works as follows. In the DFAs $M(\Lambda^i)$ of the regular domains Λ^i, for each allowed transition the output symbol 0 is added, which indicates a domain. So, while scanning a CA lattice configuration, as long as a domain is being read (i.e., allowed transitions are followed), the transducer will write a sequence of 0s. Next, the (formerly) disallowed transitions of the DFAs $M(\Lambda^i)$ are added, each with a different output symbol i that uniquely represents the corresponding transition. These added transitions generally also include transitions *between* the different DFAs, in case of more than one regular domain (or domains with multiple phases).

Using the DFAs in figure 2.6, the domain transducer for ECA 54 is constructed by first adding an output symbol 0 to the allowed transitions. In other words, the domain Λ^{54} will be mapped to 0. Next, the disallowed transitions are added, each with a unique output symbol. Note that each state in ECA 54's domain transducer has only one allowed outgoing transition, labeled with an input symbol 0 (or 1).

Consequently, each state will have one added (formerly disallowed) transition labeled with the other possible input symbol, i.e., 1 (or 0). Since there are eight states in the domain transducer, there are eight such added transitions. They can be assigned output symbols $1, \ldots, 8$, in arbitrary order. The resulting transducer is shown in figure 2.8. The allowed transitions of the original DFAs are shown with thick arrows. The added (formerly disallowed) transitions are shown with thin arrows.

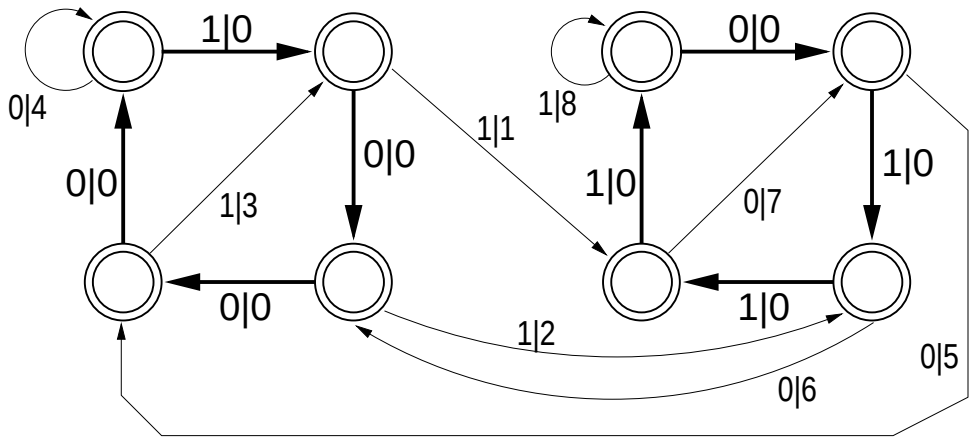

Figure 2.8: The domain transducer for ECA 54, constructed from the DFAs in figure 2.6. Added transitions are shown with thinner arrows than original transitions.

Figure 2.9 shows a simple example of how a lattice configuration of ECA 54 is mapped into a filtered configuration using the domain transducer. The lattice configuration shown in this figure consists of two repetitions of the 1110 configuration (i.e., part of a domain), followed by a 0110 configuration, followed by again two repetitions of 1110. When the domain transducer of ECA 54, as shown in figure 2.8, is used to scan this lattice configuration from left to right, the first eight symbols are mapped to 0s, since they are part of a domain. Then a violation of the domain regularity follows and the next four symbols are mapped to 5, 3, 1, and 7, respectively. The rest of the lattice consists of a domain again and is thus mapped to 0s.

Using ECA 54's domain transducer, the entire space-time diagram of figure 2.5 can be filtered this way, resulting in the space-time diagram of figure 2.10. In this figure, all occurrences of the domain Λ^{54} are mapped to white and the domain

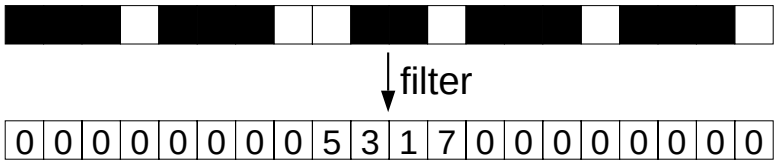

Figure 2.9: An example of the filtering process using the domain transducer.

violations are mapped to black. In other words, the output symbol 0 is represented by white cells and all output symbols $\{1, \ldots, 8\}$ in the domain transducer are represented by black cells in the filtered space-time diagram.

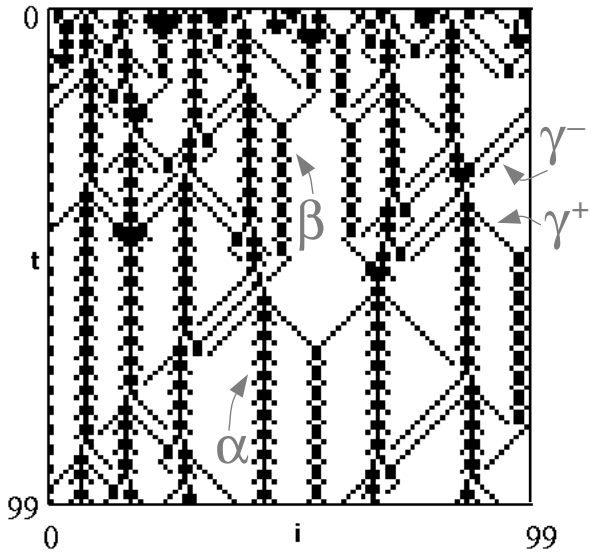

Figure 2.10: The filtered version of the space-time diagram of ECA 54 in figure 2.5.

2.4.3 Particles

The filtered space-time diagram in figure 2.10 shows where the domain regularities are violated. However, many of these violations are regular, repeating structures themselves. Thus, it seems sensible to include these structures in the CA's pattern basis too. In the CM framework, these structures are classified as particles.

A *particle* α is a spatially localized (i.e., bounded width), temporally periodic boundary between two adjacent domains. A domain-particle-domain configuration

$\Lambda^i \alpha \Lambda^j$ is also a regular language that is temporally periodic, but unlike a domain, it is not spatially homogeneous, and so it is not a regular domain itself. Particles are labeled (arbitrarily) with Greek letters. The set of all particles $\{\alpha, \beta, \ldots\}$ of a CA is denoted $\mathbf{P}$.

As a comparison of the original space-time diagram (figure 2.5) and its filtered version (figure 2.10) shows, particles can sometimes be difficult to identify directly from the raw space-time configurations. However, once the domain regularities are filtered out, the particles are revealed clearly. The filtered space-time diagram shows four different types of particles in ECA 54's dynamics. These particle types are labeled α, β, γ^+, and γ^-, after [HC97]. All four particle types form a boundary between two adjacent Λ^{54} domains. However, the domains on either side of a particle are "out of phase" with each other, either temporally or spatially or both. The different particle types are indicated with gray labels in the filtered space-time diagram in figure 2.10. Figure 2.11 shows small space-time patches with these four particles, both in original and filtered appearances.

Every particle $\alpha \in \mathbf{P}$ has a temporal periodicity p_α^t, which is the number of time steps after which it repeats itself. However, since a particle is a bounded (in space) structure, it cannot have a spatial periodicity (i.e., it cannot consist of an arbitrarily repeated spatial pattern). Therefore, when referring to the periodicity p_α of a particle α, the temporal periodicity p_α^t is implied. Thus, a particle α has p_α different phases, which can be numbered $1, \ldots, p_\alpha$. Note that each phase of a particle is characterized by a unique sequence of output symbols from the domain transducer. In other words, when a domain transducer, while reading a lattice configuration, encounters a particle, it will output a certain sequence of symbols corresponding to the state-transitions it makes, in between two sequences of 0s (domain symbols). This sequence of non-domain symbols uniquely identifies the type and phase of the particle that was just encountered.

The (temporal) periodicities of ECA 54's particles are $p_\alpha = 4$, $p_\beta = 4$, $p_{\gamma^+} = 2$,

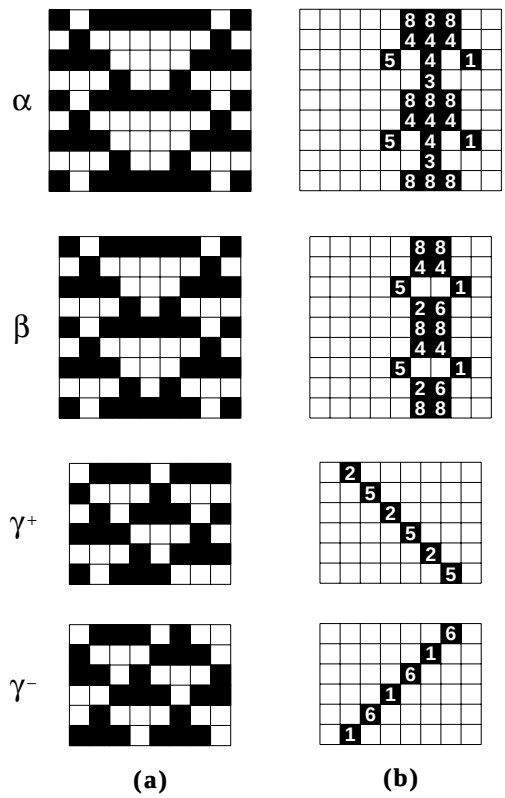

Figure 2.11: The four particles of ECA 54. (a) The unfiltered appearances. (b) The filtered appearances. Domain symbols are represented by white cells. The inscribed symbols in the black cells refer to the corresponding output symbols of the transitions in the domain transducer. After [HC97].

and $p_{\gamma^-} = 2$. Note that the particle periodicities are not directly obvious from the filtered space-time diagram in figure 2.10. For example, it seems that the γ^+ and γ^- particles have a periodicity $p = 1$. However, this is mainly a result of the fact that *all* non-domain output symbols in the domain transducer are mapped to black cells in the filtered space-time diagram. When the actual output symbols are preserved, as in figure 2.11, the periodicity $p = 2$ will clearly show. Also, the actual periodicity of a particle can be observed from the original space-time diagram. As mentioned, the different phases of each particle are identified by a unique sequence of transitions in the domain transducer while scanning a particle. These unique "signatures" can clearly be seen in figure 2.11. The four phases of the α particle, for example, are identified by the four symbol sequences '3', '888', '444', and '50401'.

In general, after a particle has completed one period, it will have shifted a number of cells in the lattice, either to the left or to the right or sometimes none at all. This number of cells a particle $\alpha \in \mathbf{P}$ has shifted after one period p_α is called its *displacement* d_α. Displacements to the left are indicated with negative numbers and displacements to the right with positive numbers. From a particle's period p and displacement d, an *average velocity* v is simply calculated as $v = d/p$.

For ECA 54, the displacement of both the α and β particles is $d = 0$, and thus their velocity is $v = 0/4 = 0$. The displacements of the γ^+ and γ^- particles are $d_{\gamma^+} = +2$ and $d_{\gamma^-} = -2$. Consequently, their velocities are $v_{\gamma^+} = +1$ and $v_{\gamma^-} = -1$.

As with the regular domains, in principle it is possible that a particle is missed when constructing the set $\mathbf{P}$ of all particles for a CA. First, if one of the domains is missed because it was never observed, its corresponding particles will also be missed (i.e., the particles that form the boundary between this domain and the other domains). Furthermore, different kinds of particles can exist as the boundary between two particular domains (or domain instances). For example, the four particles in ECA 54 all form a boundary between two Λ^{54} instances. It could be the case that there exists yet another such particle which occurs only very rarely, on very specific ICs. As with regular domains, however, for the purposes of modeling evolved CAs and predicting their performance, such rare particles do not form a significant contribution.

2.4.4 Particle interactions

As the filtered space-time diagram clearly shows, sometimes particles collide which can lead to the creation of other particles or to mutual annihilation. Such a *particle interaction* is denoted $\alpha + \beta \rightarrow \gamma$, meaning that the collision of particles α and β leads to an interaction resulting in a γ particle. An interaction result $\emptyset$ indicates an annihilative interaction. Note that the order of the interacting particles is important. The notation $\alpha + \beta$ means that the α particle is to the left of the β particle upon

collision. The complete set of possible particle interactions, given the set of particles $\mathbf{P}$, is denoted $\mathbf{I}$.

The result of a particle interaction sometimes depends on the relative phases that the interacting particles are in at the time of collision. Different relative phases at collision time can give rise to different interaction results. In that case, the particle interaction is labeled with the probability with which the interaction result occurs. For example, when two particles α and β can have two possible interaction results, say a particle γ or a particle δ, and the first result occurs 65% of the time and the second results occurs 35% of the time, then this is written as $\alpha + \beta \stackrel{0.65}{\rightarrow} \gamma$ and $\alpha + \beta \stackrel{0.35}{\rightarrow} \delta$.

It turns out that an upper bound on the number of interaction results from an interaction between two particles is given by

$$\frac{p_1 p_2 \Delta v}{p_\Lambda^t p_\Lambda^s}$$

where p_1 and p_2 are the respective periodicities of the two particles, Δv is the difference in the velocities of the two particles, and p_Λ^t and p_Λ^s are the temporal and spatial periodicities, respectively, of the domain Λ that is in between the two interacting particles before they collide. This upper bound is a generalization of the one given in [PST86] and a proof will be presented elsewhere [HSC]. So, given two particles, to find all possible interaction results from a collision between these two particles, at most $\frac{p_1 p_2 \Delta v}{p_\Lambda^t p_\Lambda^s}$ possibilities need to be checked.

Given the set $\mathbf{P}$ of four particle types, the following particle interactions can be observed in ECA 54:

$$\begin{array}{lcl}
\alpha + \gamma^- & \rightarrow & \gamma^- + \alpha + 2\gamma^+ \\
\gamma^+ + \alpha & \rightarrow & 2\gamma^- + \alpha + \gamma^+ \\
\beta + \gamma^- & \rightarrow & \gamma^+ \\
\gamma^+ + \beta & \rightarrow & \gamma^- \\
\gamma^+ + \gamma^- & \rightarrow & \beta \\
\gamma^+ + \alpha + \gamma^- & \rightarrow & \gamma^- + \alpha + \gamma^+ \\
\gamma^+ + \beta + \gamma^- & \rightarrow & \emptyset
\end{array}$$

For ECA 54, there are no particle interactions that can have more than one interaction result. The expression for the upper bound on interaction results comes out to 1 for all possible particle interactions. Note also that the last two interactions are three-particle interactions. Most of these interactions can be observed in figure 2.10.

Note that the words *particle* and *particle interaction* are purely used as an analogy here. The reason these structures are called particles is because they behave somewhat similar to physical particles. For example, the filtered space-time diagram in figure 2.10 is reminiscent of a picture of particles in a bubble chamber. The particles in a CA travel with a certain velocity and interact with each other, annihilating or creating other particles, just like physical particles. However, the analogy should not be taken too far. For example, there is generally no equivalent in the particles in a CA of preservation of momentum or energy as in physical particles. So, the name *particle* implies an analogy, not a one-to-one mapping.

2.4.5 The particle catalog

The complete set $\{\mathbf{\Lambda}, \mathbf{P}, \mathbf{I}\}$ of domains, particles, and particle interactions, i.e., the pattern basis of a CA, can be summarized in a *particle catalog*. This catalog forms a description, expressed in computation theoretic ways, of the CA's dynamics at a higher level than the CA update rule ϕ itself, or even the raw space-time configurations. Furthermore, this pattern basis is identified and classified without necessarily assigning semantics or usefulness to the discovered patterns (i.e., the domains and particles). In this sense, it reveals the intrinsic computation embedded in the dynamics of a CA [Cru94a].

The particle catalog for ECA 54 is given in table 2.1. Note that only the stable domain Λ^{54} is included in the set $\mathbf{\Lambda}$, but not the other known but unstable domains. In [HC97], an analysis is done of how much of the CA lattice can be described in terms of the pattern basis summarized here in table 2.1. These results indicate that, after a short transient time, only about 1% of the entire lattice is not captured by

this higher level description of ECA 54's dynamics.

Domains Λ					**Interactions I**		
Label	Regular language				$\alpha + \gamma^-$	$\rightarrow$	$\gamma^- + \alpha + 2\gamma^+$
Λ^{54}	$(0001)^*, (1110)^*$				$\gamma^+ + \alpha$	$\rightarrow$	$2\gamma^- + \alpha + \gamma^+$
Particles P					$\beta + \gamma^-$	$\rightarrow$	γ^+
Label	Boundary	p	d	v	$\gamma^+ + \beta$	$\rightarrow$	γ^-
α	$\Lambda^{54}\Lambda^{54}$	4	0	0	$\gamma^+ + \gamma^-$	$\rightarrow$	β
β	$\Lambda^{54}\Lambda^{54}$	4	0	0	$\gamma^+ + \alpha + \gamma^-$	$\rightarrow$	$\gamma^- + \alpha + \gamma^+$
γ^+	$\Lambda^{54}\Lambda^{54}$	2	2	1	$\gamma^+ + \beta + \gamma^-$	$\rightarrow$	$\emptyset$
γ^-	$\Lambda^{54}\Lambda^{54}$	2	-2	-1			

Table 2.1: The particle catalog of ECA 54.

Particles and their interactions can be interpreted as encoding, transmitting, and processing information, as is shown in the next chapter. The seemingly complex global behavior of a CA can be broken down into these relatively simple patterns, or information units, which provides a concise description of this global behavior. The computational mechanics framework thus begins to answer the last problem from "Twenty problems in the theory of cellular automata" [Wol85]: *What higher-level descriptions of information processing in cellular automata can be given?*

Chapter 3

Evolving Cellular Automata with Genetic Algorithms

The main motivation behind the *evolving cellular automata* (EvCA) framework is to understand how genetic algorithms evolve cellular automata that perform computational tasks requiring global information processing. Since the individual cells in a CA can communicate only locally, without the existence of a central control, the GA has to evolve CAs that exhibit higher-level, emergent behavior in order to perform this global information processing. Thus, this framework provides an approach to studying how evolution can create dynamical systems in which the interactions of simple components with local information storage and communication give rise to coordinated global information processing. The current chapter first reviews previous work on evolving CAs to perform certain tasks, and then presents new results on evolving CAs on additional tasks.

3.1 Cellular automata implementation

The experiments described in the next several sections all involve one-dimensional two-state ($k = 2$) CAs with radius $r = 3$ and periodic boundary conditions. Unless mentioned otherwise, a lattice of size $N = 149$ or $N = 150$, depending on the task, is used, so that $N \gg r$. The CA alphabet is $\Sigma = \{0, 1\}$, where 0s are represented by white cells and 1s by black cells in space-time diagrams.

A $(k, r) = (2, 3)$ CA has $k^{2r+1} = 2^7 = 128$ entries in the lookup table representing the local update rule ϕ. This lookup table is represented as a bit string of length 128, assuming a lexicographical ordering of the local neighborhood configurations η. In other words, the first bit in the string is the output bit s_{t+1} for the local neighborhood $\eta = 0000000$, the second bit for $\eta = 0000001$, etc., until the last bit for $\eta = 1111111$.

Since there are 128 entries in the lookup table, each of which can be either 0 or 1, there are $2^{128} \approx 10^{38}$ $(k, r) = (2, 3)$ CAs. Of course, this space of possible CA rules is far too large to search exhaustively.

3.2 Computational tasks for cellular automata

As mentioned in the introduction, several kinds of computation in CAs can be distinguished. For the computational tasks considered here, a CA performing a computation will mean that the input to the computation is encoded as the IC $\mathbf{s}_0$, the output is decoded from the lattice configurations $\mathbf{s}_{M+i},\ i \geq 0$, after some number of time steps $M \gg 0$, and the intermediate configurations $\mathbf{s}_t$, $0 < t < M$, that transform the input to the output are taken as the steps in the intervening computation. Figure 3.1 illustrates this notion of computation in CAs.

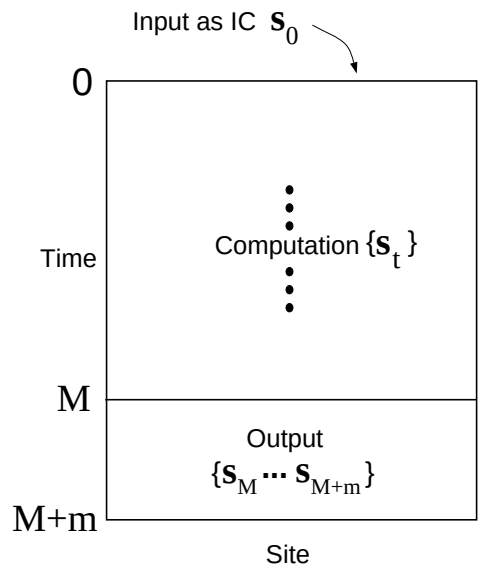

Figure 3.1: Schematic representation of computation in CAs as used here.

A *computational task* $\mathbf{T}$ for a CA is now defined as a mapping from initial configurations $\mathbf{s}_0$ to sets of answer states $\mathcal{A}_i, 1 \leq i \leq n$, for some finite n. An answer state $\mathcal{A}_i$ consists of a finite set of m configurations $\mathbf{s}_j, 1 \leq j \leq m$, which is time invariant ($m = 1$) or through which the CA must cycle repeatedly ($m \geq 2$). Formally,

$$\mathbf{T} : \Sigma^N \rightarrow \{\mathcal{A}_i : 1 \leq i \leq n\}$$

such that

$$\mathbf{T}(\mathbf{s}_0) = \begin{cases} \mathcal{A}_1 & \text{if } \mathbf{s}_0 \in \mathcal{C}_1 \\ \mathcal{A}_2 & \text{if } \mathbf{s}_0 \in \mathcal{C}_2 \\ & \dots \\ \mathcal{A}_n & \text{if } \mathbf{s}_0 \in \mathcal{C}_n \end{cases}$$

where the $\mathcal{C}_i$, $1 \leq i \leq n$, form a partition of $\{0,1\}^N$, i.e., $\mathcal{C}_i = \mathcal{C}_j \iff i = j$, $\mathcal{C}_i \cap \mathcal{C}_j = \emptyset, i \neq j$, and $\bigcup_{i=1}^{n} \mathcal{C}_i = \{0,1\}^N$. If the IC $\mathbf{s}_0 \in \mathcal{C}_i$, then the CA has to settle down into answer state $\mathcal{A}_i$. The maximum number of time steps M by which the CA has to reach this answer state is a function of the lattice size N.

Given a task $\mathbf{T}$ and a CA Φ, the *performance* $\mathcal{P}_N$ of Φ on $\mathbf{T}$ on lattice size N is defined as the fraction of initial configurations $\mathbf{s}_0$ on which Φ settles down to the correct answer state $\mathcal{A}_i$ within M time steps (i.e., if $\mathbf{s}_0 \in \mathcal{C}_i$, then $\mathbf{s}_{M+j} \in \mathcal{A}_i, j \geq 0$). Since there are $|\Sigma|^N$ possible ICs on a lattice of size N, it becomes practically impossible to calculate a CA's performance on *all* ICs for large values of N. Therefore, a CA's performance $\mathcal{P}_N$ is generally approximated by calculating it over a smaller, but representative random sample of ICs.

Note that the CA is given a maximum number M of time steps to settle down into an answer state. If the CA has not reached an answer state within M time steps on a particular IC, it is counted as an incorrect answer, even though the CA might still settle down to the correct answer state after an additional number of time steps. In other words, not only does the CA have to give the correct answer, it also has to do so in a limited number of time steps. This is equivalent to putting a finite upper limit on the number of steps that, say, a Turing machine is allowed to do to return an answer given some input. As a consequence, no equivalent of a halting state is necessary. The system is run for a fixed number of steps after which the answer is checked.

The two computational tasks that have been studied the most extensively in previous EvCA work are density classification and global synchronization. Both these

tasks require global information processing and, as explained shortly, are non-trivial for a CA. In addition, two new tasks, variants of the original global synchronization task, are studied in this dissertation. These four tasks are explained next.

3.2.1 Density classification

For the *density classification* task $\mathbf{T}_{dens}$, the goal is to find a CA that decides whether or not the initial configuration $\mathbf{s}_0$ contains a majority of 1s (i.e., has high density). Let $\rho_0 = \rho(\mathbf{s}_0)$ denote the density of 1s in the IC. If $\rho_0 > 0.5$, then within M time steps the CA should reach the fixed-point configuration of all 1s (i.e., all cells in state 1 for all subsequent iterations); otherwise, within M time steps it should reach the fixed-point configuration of all 0s.

More formally, the density classification task can be described as follows. There are $n = 2$ classes of ICs: $\mathcal{C}_1 = \{\mathbf{s}_0 : \rho(\mathbf{s}_0) < 0.5\}$ and $\mathcal{C}_2 = \{\mathbf{s}_0 : \rho(\mathbf{s}_0) > 0.5\}$. Within each of the two corresponding answer states, there is $m = 1$ configuration: $\mathcal{A}_1 = \{0^N\}$ and $\mathcal{A}_2 = \{1^N\}$. So, the density classification task is a mapping

$$\mathbf{T}_{dens} : \{0,1\}^N \rightarrow \{\{0^N\}, \{1^N\}\}, \quad N \text{ odd}$$

which is defined as:

$$\mathbf{T}_{dens}(\mathbf{s}_0) = \begin{cases} \{0^N\} & \text{if } \rho(\mathbf{s}_0) < 0.5 \\ \{1^N\} & \text{if } \rho(\mathbf{s}_0) > 0.5 \end{cases}$$

Note that N odd is used so that the IC classes $\mathcal{C}_1$ and $\mathcal{C}_2$ unambiguously partition Σ^N (i.e., $\rho_0 = 0.5$ cannot occur). In the experiments reviewed below, $N = 149$ is used.

This task is clearly non-trivial for any CA with $r \ll N$ since the density of the IC is a global property of the lattice, while each cell in the lattice can communicate only with its $2r$ direct neighbors. The minimum amount of memory required for this task is $\mathcal{O}(\log(N))$, since the equivalent of a counter register is needed to keep track of the excess of 1s in a serial scan of the IC. However, checking a CA's answer state at time step M requires only $\mathcal{O}(1)$ memory. In other words, the CA is asked

to implement a mapping from a non-regular language (in particular, a context-free language) to a regular language.

It has been shown that no two-state finite radius CA can perform this task perfectly on all (odd) lattice sizes [LB95, Das98]. However, it is not known what the maximum performance of a CA on this task can be.

3.2.2 Global synchronization–1

For the original global synchronization task, called global synchronization–1 here, or $\mathbf{T}_{sync1}$, the goal is to find a CA that, from any IC, settles down within M time steps to a temporally periodic oscillation between an all-0s configuration and an all-1s configuration. So, for this task there is only one class of ICs, namely all possible ICs: $\mathcal{C}_1 = \{\Sigma^N\}$. The answer state corresponding to $\mathcal{C}_1$ consists of two configurations: $\mathcal{A}_1 = \{0^N, 1^N\}$, with the additional constraint that $\Phi(0^N) = 1^N$ and $\Phi(1^N) = 0^N$. Thus, the global synchronization–1 task is the following mapping:

$$\mathbf{T}_{sync1} : \{0,1\}^N \rightarrow \{\{0^N, 1^N\}\}$$

In other words, after at most M time steps, the CA has to cycle repeatedly through the configurations 0^N and 1^N, giving rise to alternating white and black horizontal stripes in the space-time diagrams. As with the density classification task, a lattice of size $N = 149$ is used for this task.

This task is non-trivial for any CA with $r \ll N$ since the required synchronous oscillation is a global property of the lattice, i.e., all cells in the lattice have to be in the same state at the same time step, and change states simultaneously.

3.2.3 Global synchronization–2

The global synchronization–2 task $\mathbf{T}_{sync2}$ is a variation on $\mathbf{T}_{sync1}$. Instead of temporally periodic horizontal stripes, the requirement is spatially periodic vertical stripes.

More formally, the answer state for this task is $\mathcal{A}_1 = \{(01)^{N/2} + (10)^{N/2}\}$; the white vertical stripe may be either on the even cells or on the odd cells in the lattice. So,

$$\mathbf{T}_{sync2} : \{0,1\}^N \rightarrow \{\{(01)^{N/2} + (10)^{N/2}\}\}$$

where $\mathbf{s}_{t+1} = \mathbf{s}_t,\ t \geq M$. Note that for this task an even lattice size N is needed, otherwise the required pattern will not fit on the lattice. Here, $N = 150$ is used.

As with the original global synchronization task, this variant is non-trivial for a CA since the required synchronized pattern is a global property of the lattice. Either all even cells have to be white and all odd cells have to be black, or vice versa.

3.2.4 Global synchronization–3

The global synchronization–3 task $\mathbf{T}_{sync3}$ is a combination of both $\mathbf{T}_{sync1}$ and $\mathbf{T}_{sync2}$. The required pattern is a so called "checkerboard", or alternating white and black cells both in space and in time. The one answer state consists of 2 configurations: $\mathcal{A}_1 = \{\mathbf{s}_1, \mathbf{s}_2\} = \{\{(01)^{N/2} + (10)^{N/2}\}, \{(10)^{N/2} + (01)^{N/2}\}\}$, where it is assumed that if $\mathbf{s}_1 = (01)^{N/2}$, then $\mathbf{s}_2 = (10)^{N/2}$ and vice versa. In other words, $\Phi((01)^{N/2}) = (10)^{N/2}$ and $\Phi((10)^{N/2}) = (01)^{N/2}$. So,

$$\mathbf{T}_{sync3} : \{0,1\}^N \rightarrow \{\{(01)^{N/2} + (10)^{N/2}\}, \{(10)^{N/2} + (01)^{N/2}\}\}$$

As for $\mathbf{T}_{sync2}$, the $\mathbf{T}_{sync3}$ task requires an even lattice size. Again, $N = 150$ is used. Since this task is a combination of the previous two global synchronization tasks, it is obviously also non-trivial for a finite-radius CA.

3.3 Genetic algorithms

Genetic algorithms (GAs) [Hol75, Gol89, Mit96] are a class of stochastic search methods inspired by biological evolution. Instead of stating *how* to solve a given problem, a GA is implicitly (through a feedback mechanism) told *what* to solve. GAs are widely

used to solve difficult problems and to study evolutionary phenomena. In the evolving cellular automata framework, they are used to evolve CAs to perform computational tasks. In this section, first a brief overview of a general genetic algorithm is given, and then the particular GA implementation and parameter values used in the EvCA framework are presented.

3.3.1 The general algorithm

A GA tries to find a satisfactory answer to some given problem by a process of simulated evolution. To do this, the GA maintains a *population* of candidate solutions. Usually, these candidate solutions are represented as bit strings (*genotypes*) that encode solutions (*phenotypes*) to the given problem. Each individual (bit string) in the population is assigned a *fitness* value, which reflects how well the encoded solution solves the given problem. In other words, the better a solution it represents, the higher an individual's fitness value will be. These fitness values are assigned by a *fitness function*, which takes as input a bit string, decodes it to the corresponding solution, tests this solution on the given problem, and returns a fitness value according to how well the solution actually solves the problem.

Next, individuals from the population are selected at random, with a probability proportional to their fitness values, and placed in a *mating pool*. The higher an individual's fitness is relative to the rest of the population, the higher its chance of being selected (perhaps even more than once, since this selection is generally done with replacement). And, of course, the lower an individual's fitness, the lower its chance of being selected (such an individual might not even get selected at all).

Once the mating pool is filled with (sometimes multiple) copies of individuals from the old population, a new population is created with offspring of individuals from this mating pool. To do this, generally two *parents* are chosen from the mating pool, at random or according to some other scheme. Then, with a certain probability p_c, *crossover* is applied, in which the "genetic" information of the two parents is

mixed, resulting in two *children*. Finally, with a usually small probability p_m, some of the bits in the newly created children are *mutated*, i.e., a 0 is turned into a 1 or vice versa. Then this offspring is placed in the new population. This process is repeated until the new population is filled, after which the fitness of the individuals in this new population are re-evaluated.

This sequence of fitness assignments, selection, crossover, and mutation is repeated over many *generations*, usually starting with a random initial population. The idea is that better and better (i.e., more fit) individuals are *evolved* over time by combining partial solutions (for example via crossover) and making small improvements (via mutations). Changes in genetic information caused by crossover and mutation that actually decrease an individual's fitness value will be discarded and changes that cause an increase in fitness will be preserved by the selection process.

GAs are widely used to solve (numerical) optimization problems that have many variables that need to be optimized. They have been applied successfully to find (near) optimal solutions, and are known to frequently outperform standard solution techniques for certain problems. Furthermore, GAs have been successfully used to model and explain certain phenomena that occur in natural evolution. For an extensive overview of all these applications of GAs, see the various GA conference proceedings [Gre85, Gre87, Sch89, BB91, For93, Esh95, Bäc97].

3.3.2 Implementation and parameter values

Of course, a GA can be implemented with many variations: different types of representations (bit strings, permutations, real numbers, etc.), different selection operators (fitness-proportionate selection, elitist selection, rank selection, etc.), different kinds of crossover (one-point, multi-point, uniform, etc.), and different kinds of mutations (depending on the representation used). For example, see [Dav91] for an overview of some of these different implementations and how they might improve the GA search performance. Additionally, given one particular implementation, different values for

the respective parameters in the algorithm can be used (population size, selection strength, crossover probability p_c, mutation rate p_m, and so on). This section gives an overview of the GA implementation and parameter values used in the evolving cellular automata framework.

The population

The GA population consists of bit strings representing $(k, r) = (2, 3)$ CA lookup tables. Since these lookup tables have 128 entries (see section 3.1), the individuals in the GA population are strings of 128 bits. A lexicographical ordering of the lookup table entries is assumed. A population size P of 100 is used. In the following, the words "individual", "bit string", "CA rule", and "lookup table" are used interchangeably, but they all refer to members of the GA population representing CA rules.

The initial population of bit strings in the GA is created at random. In the EvCA experiments, this is done in one of two ways. The first method is to randomly assign a 0 or a 1 (with equal probability) to each position in each of the bit strings. This creates a population of CA rules with a binomial distribution of λ values around 0.5 (recall that the λ value of a two-state CA is simply the fraction of 1s in the output states of the lookup table). In other words, the fraction of 1s in the lookup tables of these randomly created CA rules is clustered around $\lambda = 0.5$.

The second method is to create a population of CA rules with a uniform distribution of λ over the interval $[0, 1]$. In this case, there are some CA rules in the initial population with very few 1s, some with an intermediate number of 1s, and some with very many 1s. To achieve this, the interval $[0, 1]$ is divided up into 20 density bins of equal length. For each bin, 5 CA rules are created at random with a λ value that falls within that particular density bin.

The fitness function

The fitness function, of course, depends on the particular task for which the CAs are being evolved. Given a task $\mathbf{T}$, with IC classes $\mathcal{C}_i$ and corresponding answer states $\mathcal{A}_i$, the fitness function works as follows. First, the input to the fitness function, which is a bit string of length 128, is converted into a CA lookup table using the convention that the lookup table entries are lexicographically ordered. This CA is then run on a test set of 100 randomly created ICs. The CA's fitness then is the fraction of these 100 ICs on which it settles down, within M time steps, to the correct answer state. The maximum number of iterations M for the CA is set to $M = 2N$, i.e., twice the lattice size.

As an example, take the density classification task $\mathbf{T}_{dens}$. Given a CA Φ, this CA is run on 100 random ICs. For each IC for which $\rho_0 < 0.5$ and the CA correctly settles down to the configuration 0^N within M time steps, the CA receives a score of 1. Similarly for $\rho_0 > 0.5$ and 1^N. If for some IC the CA settles down to the wrong answer state, or fails to settle down to any answer state at all, it gets a score of 0 for that particular IC. The fitness of the CA is then calculated as the total score (over all ICs) divided by the total number of ICs (100 in this case).

The test set of 100 ICs is created in one of two ways: an "unbiased" or a "biased" method. In the unbiased method, 0s and 1s are randomly assigned with equal probability. This generates a binomial distribution of initial densities ρ_0 around 0.5. In other words, the number of 1s in the ICs is clustered around 0.5. In the biased method, 0s and 1s are assigned according to a uniform distribution over ρ_0. In other words, there are some ICs with very few 1s, some with an intermediate number of 1s, and some with very many 1s. Note that these two ways of generating ICs are equivalent to the two ways of generating random CA rules for the initial GA population.

Finally, it should be mentioned that during a GA run, a new set of 100 random ICs is generated at the beginning of each generation, and the fitness values of all 100

CA rules in the population in that generation are calculated using this new set of ICs. In other words, in one particular generation all CAs in the population are tested on the same set of ICs, but this set is different in each generation.

The selection operator

The selection operator used here is called *elitist* selection. Under elitist selection, in each generation the population is sorted according to fitness in decreasing order. Then the top E members of the population are selected and copied without modification to the new population. In other words, in each generation the E best individuals in the population survive, and the other $P - E$ individuals "die".

The remaining $P-E$ "slots" in the new population are filled with offspring from the E elite CA rules. Each time, two parents out of the E elite are selected at random. Crossover is then applied to create two children, which are subjected to mutation. The offspring are then placed in the new population. This process is repeated until the new population is filled (i.e., until it contains a total of P individuals).

In the experiments reported here, an elite size of $E = 20$ is used.

The crossover operator

Since crossover is always applied between two randomly selected parents from the elite, the crossover probability is $p_c = 1$. The crossover operator that is used here is *one-point* crossover. In this operator, a random crossover point is chosen somewhere between the first and the last bit of two parents a and b. The parts of the bit strings after this crossover point are then swapped between the two parents, creating two children a' and b' that contain "genetic material" from both parents. Figure 3.2 shows one-point crossover schematically.

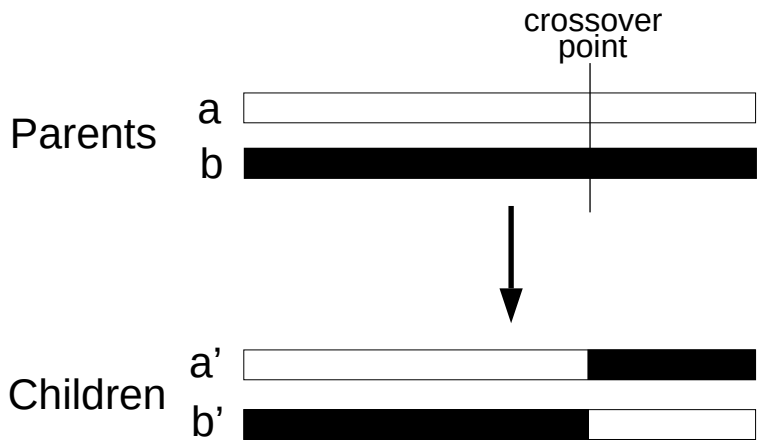

Figure 3.2: The one-point crossover operator. The parts of the bit strings after a randomly chosen crossover point are swapped between the parents a and b, creating two children a' and b'.

The mutation operator

A mutation rate of $p_m = 0.016$ per bit is used. This means that every bit in a newly created individual has a probability of 0.016 of being flipped, i.e., a 0 is turned into a 1 or vice versa. With this mutation rate, on average $0.016 \times 128 = 2$ bits per newly created individual are flipped.

The number of generations

Each run of the GA is done for $G = 100$ generations. Occasionally, a particular GA run is continued for another 100 generations. This is done, for example, to see if an interesting looking but not yet optimal solution that is present in generation 100 can still be improved on.

Summary of GA parameters

Table 3.1 summarizes the GA operators and parameter settings used in the evolving cellular automata framework.

Population	bit string of length 128
	population size $P = 100$
Fitness function	fraction of correct answers on random test IC sets of size 100
	maximum number of CA iterations $M = 2N$
Selection	elitist selection, $E = 20$
Crossover	one-point, $p_c = 1.0$
Mutation	$p_m = 0.016$ (i.e., 2 bits flipped on average)
Generations	$G = 100$

Table 3.1: The GA operators and parameters setting used in the EvCA framework.

3.4 The evolution of emergent computation in cellular automata

The first work on evolving cellular automata with genetic algorithms was done by Packard [Pac88]. Evolving one dimensional $(k, r) = (2, 3)$ CAs to perform the density classification task, he studied the frequency of CA rules in the population as a function of Langton's λ parameter. The results showed two distinctive peaks in the frequency distribution around critical λ values in the final generations of the GA. Recall that Langton had associated these critical λ values with a phase transition in CA behavior between periodic and chaotic, claiming that complex behavior (Wolfram's class 4) occurs at those λ values. Packard interpreted his own results as supporting the following two hypotheses: (1) CAs that are capable of performing complex computations are most likely to be found near critical λ values, and (2) when CAs are evolved to perform a complex computation, evolution will tend to select CA rules with λ values close to the critical value.

In an independent series of experiments, Mitchell and colleagues tried to replicate Packard's results [MHC93]. However, they were unable to reproduce the results and, on the contrary, found that the CA rules evolved towards λ values *away* from the critical values. Using theoretical arguments and these empirical results, they showed

that useful computation in CAs does not necessarily occur near critical λ values, and that the results depend on the particular computational task that the CAs are evolved for.

Mitchell and colleagues continued with a more thorough investigation of the evolution of CAs for the density classification task, and also studied the dynamics that gave rise to the computational capabilities of the evolved CAs [MHC93, MCH94a, MCH94b]. However, in this first series of experiments no sophisticated emergent computational strategies in the CAs were evolved by the GA, and [MCH94b] discusses a number of impediments that prevented the GA from finding such strategies.

In a subsequent series of experiments, however, CAs with sophisticated emergent computational strategies were actually found in a small percentage of the GA runs, both on the density classification task [DMC94, CM95] and on the global synchronization–1 task [DCMH95]. Both the evolutionary history and the emergent strategies of these evolved CAs were analyzed in more detail in [Das98, CMD98]. The main results of these experiments are reviewed briefly here.

Figure 3.3 shows the result of a particular run of the GA on the density classification task $\mathbf{T}_{dens}$. This result was first presented in [DMC94]. The first graph in figure 3.3 shows the best fitness in the population over time during the GA run. The fitness of a CA in this run was calculated over 100 random ICs, generated with a uniform distribution over ρ_0 (see section 3.3.2). Note that the fitness starts out in generation 0 at $f = 0.5$. The reason for this is that the initial population in the GA was created with a uniform distribution over the λ values of the CAs. This means that there will be some CAs in the initial population that have (almost) all 0s in their lookup table. Such a CA will quickly settle down to an all-0s configuration, since (almost) all neighborhood configurations are mapped to a 0. Furthermore, since the ICs are created at random, half of them will have low density (i.e., $\rho_0 < 0.5$). Thus, a CA with (almost) all 0s in its lookup table will correctly classify the 50% of ICs with low density and incorrectly classify the other 50% of high density, and

consequently have a fitness of 0.5. Similarly, there will be some CA rules in the initial population with (almost) all 1s in the lookup table, which quickly settle down to an all-1s configuration and consequently also have a fitness of 0.5.

After about 10 generations, however, the best fitness value in the population suddenly jumps up from the 0.5 fitness level, and from there on increases quickly. The arrows (labeled ϕ_{dens1} to ϕ_{dens5}) indicate generations in which a significant improvement in fitness occurred. The space-time diagrams in figure 3.3 show the typical behaviors of the best CAs in the population in those generations. These are the CAs that gave rise to these increases in fitness.

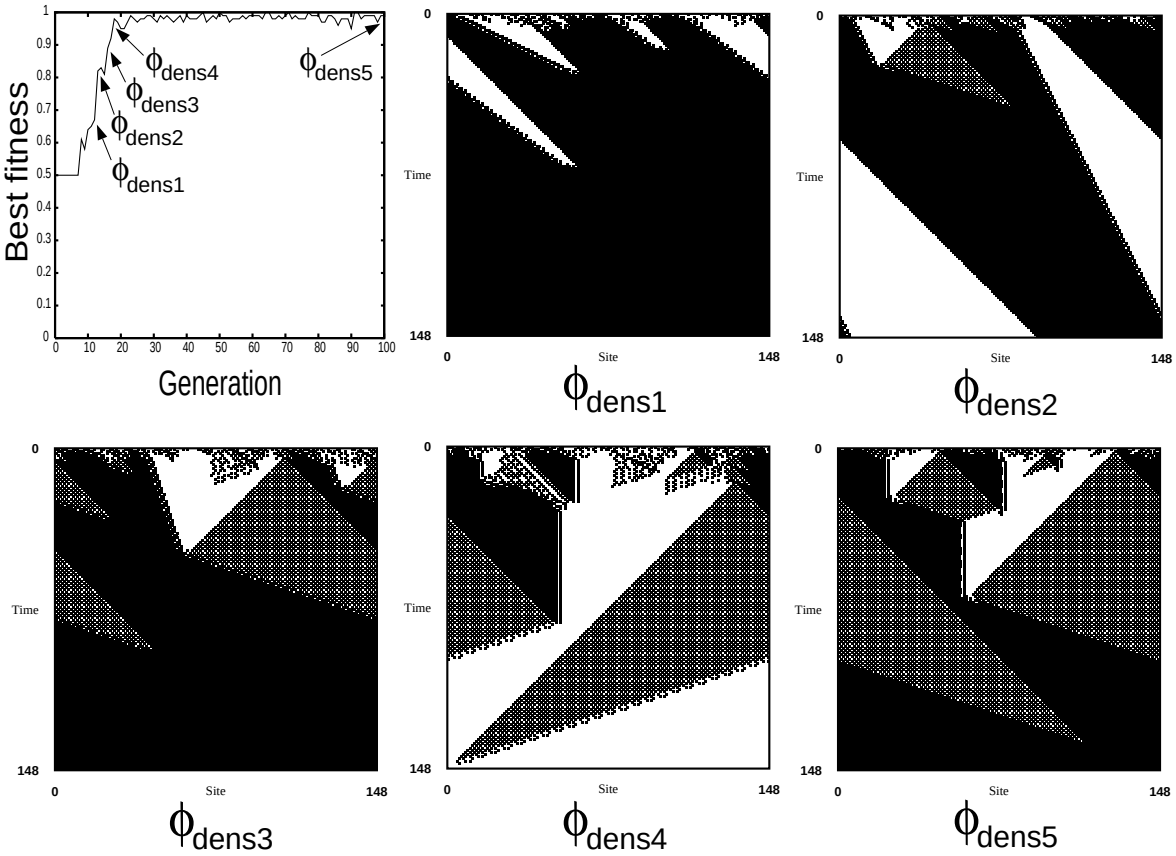

Figure 3.3: The result of a particular GA run on the density classification task. The best fitness versus generation is plotted. Space-time diagrams of CAs that gave rise to significant improvements in fitness during the evolution are shown. After [DMC94].

It is clear that the evolved CAs, after an initial transient behavior, quickly

settle down into local homogeneous regions of either all-0s (white), all-1s (black), or alternating 0s and 1s ("checkerboard"), with propagating boundaries between these regions. Roughly, the "strategy" used by ϕ_{dens5}, the best CA in the final generation, can be described as size competitions between these local regions, occurring at different time and length scales, where the largest regions eventually win over smaller regions. This strategy, along with those of the other CAs shown, is analyzed in more detail in the next section using the computational mechanics framework.

Figure 3.4 shows the results of a particular run on the original global synchronization task $\mathbf{T}_{sync1}$. This result was first presented in [DCMH95]. The first graph in figure 3.4 shows the best fitness over time. Again, the fitness was measured over 100 random ICs with a uniform distribution over ρ_0. The arrows mark generations in which a significant increase in fitness occurred. The space-time diagrams of the corresponding CAs show their typical behaviors.

Again, the CAs quickly settle down, after an initial transient period, into local homogeneous regions of synchronized patterns or regions with "zig-zag" patterns, with propagating boundaries between them. As with the density classification task, the strategy of, say, ϕ_{sync5}, the best CA in the final generation, can be explained as size competitions between these local regions, eventually resolving any phase differences that exist between the different locally synchronized regions. These CAs too, are analyzed in more detail in the next section using the computational mechanics framework.

3.5 The analysis of evolved cellular automata

Since the notions of "homogeneous regions" and "propagating boundaries" between these regions are formalized as regular domains and particles in computational mechanics for CAs, this framework forms an appropriate tool for analyzing the emergent behavior and the evolutionary history of the CAs that were evolved by the GA. As

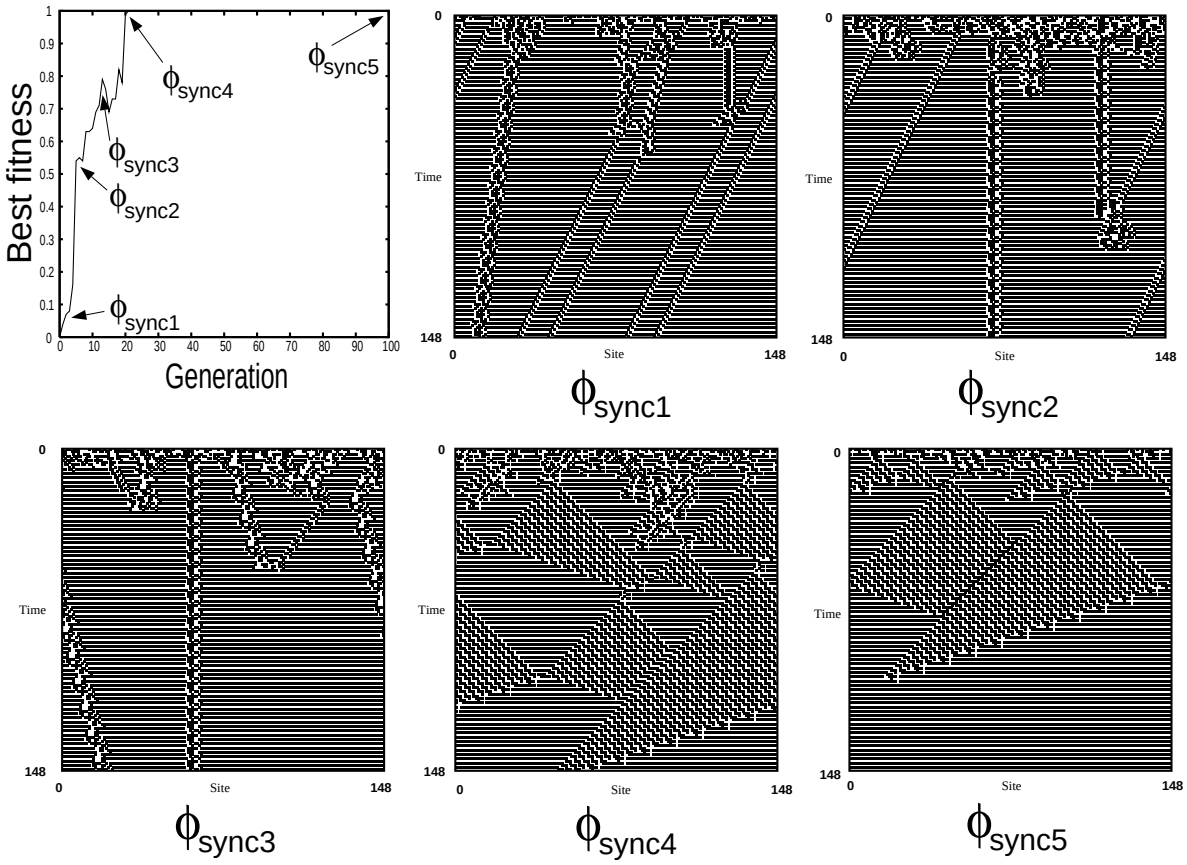

Figure 3.4: The result of a particular GA run on the original global synchronization task. The best fitness versus generation is plotted. Space-time diagrams of CAs that gave rise to significant improvements in fitness during the evolution are shown. After [DCMH95].

shown in chapter 2, filters can be built that suppress the domains, revealing the particles and their interactions more clearly. Furthermore, a particle catalog can then be constructed, containing the relevant information about the domains, particles, and particle interactions. The *computational strategies* of the evolved CAs, i.e., the strategies they use for performing the given computational task, can then be described in terms of these particle catalogs.

As an example, consider ϕ_{sync5}, the best CA in the final generation of the GA run shown in figure 3.4 above. Two distinct patterns can be observed in this CA's behavior: (1) the synchronization pattern and (2) the "zig-zag" pattern. The

first pattern can be represented by the set of regular expressions $\Lambda^s = \{0^*, 1^*\}$ and the second pattern by $\Lambda^z = \{(0001)^*, (1110)^*\}$. The corresponding (minimal) DFAs are shown in figure 3.5. These DFAs are clearly strongly connected. Furthermore, it can be shown, using these DFAs, the update transducer $T_{\Phi_{\text{sync5}}}$, and the FME algorithm, that these patterns are mapped onto themselves by the dynamics of ϕ_{sync5}. Consequently, both Λ^s and Λ^z are regular domains of ϕ_{sync5}.

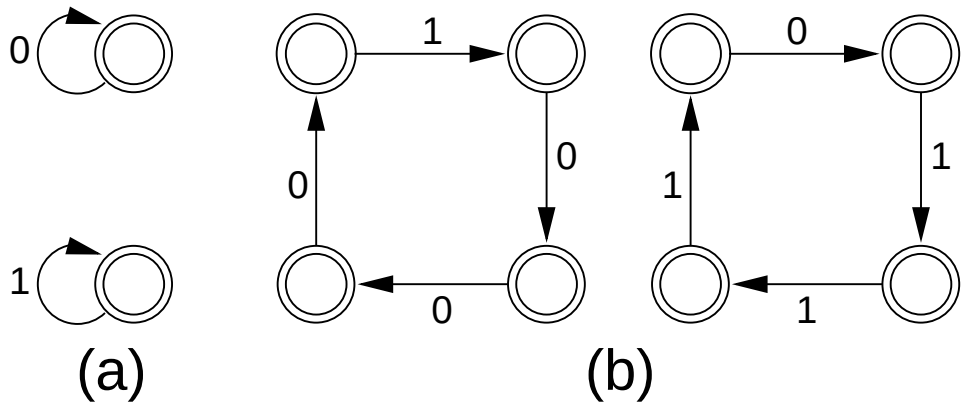

Figure 3.5: The DFAs representing the regular domains of ϕ_{sync5}. (a) $\Lambda^s = \{0^*, 1^*\}$ and (b) $\Lambda^z = \{(0001)^*, (1110)^*\}$.

Note that the regular domain $\Lambda^z = \{(0001)^*, (1110)^*\}$ of ϕ_{sync5}, expressed as a regular expression, is equal to the regular domain Λ^{54} of ECA 54. However, the actual pattern in their respective space-time diagrams is different. This illustrates the ambiguity in the regular expression representation of a domain, as mentioned in section 2.4.1. The temporal phases of the two domains Λ^z and Λ^{54} are indeed the same, but the difference in the actual appearance of these domains in a space-time diagram is due to the difference in the way in which these temporal phases are mapped into each other by the respective CA dynamics.

In section 2.4.1, this dynamical relation between the two temporal phases of Λ^{54} was shown by adding "temporal transitions" to the DFAs representing the two phases of this domain (see figure 2.7). Doing this for the DFAs representing the two temporal phases of Λ^z results in the automaton shown in figure 3.6. This "space-

time" automaton is indeed different from that in figure 2.7 (it looks the same at first glance, but note the that inner DFA is oriented differently and that the temporal transitions are directed in the opposite way, compared to figure 2.7).

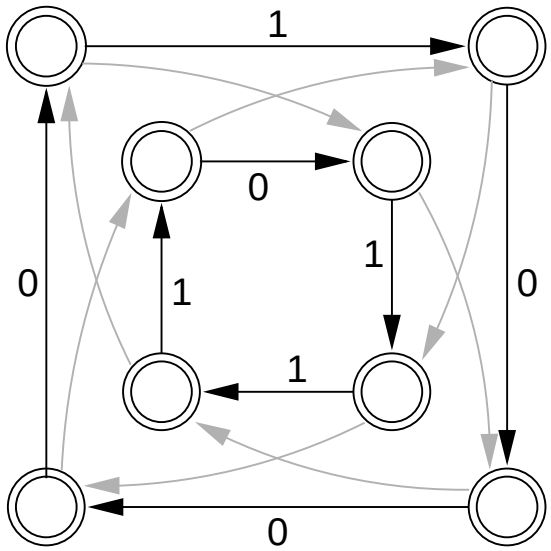

Figure 3.6: The minimal DFAs of figure 3.5, representing Λ^z, with the temporal transitions added in gray.

Using the (original) DFAs of figure 3.5, the domain transducer for ϕ_{sync5} can be constructed. This transducer is then used to filter out the domains in the space-time behavior of ϕ_{sync5}. Figure 3.7 shows an example. In figure (a) a space-time diagram of ϕ_{sync5} is shown and in figure (b) the filtered version is shown. The different particles are labeled with Greek letters.

Characterizing the particles and their interactions this way, the relevant information about ϕ_{sync5}'s emergent behavior can be collected in a particle catalog. Table 3.2 shows the complete catalog for ϕ_{sync5}. The notations Λ^s_0 and Λ^s_1 are used to distinguish the two different phases of the domain Λ^s; similarly for the Λ^z domain. Note that there is one particle interaction that can have two different results. The probabilities of the different outcomes are simply determined by counting, over a large set of random ICs, the fraction of occurrences of these interaction results.

The same computational mechanics analysis can be done for the CAs ϕ_{dens1} to

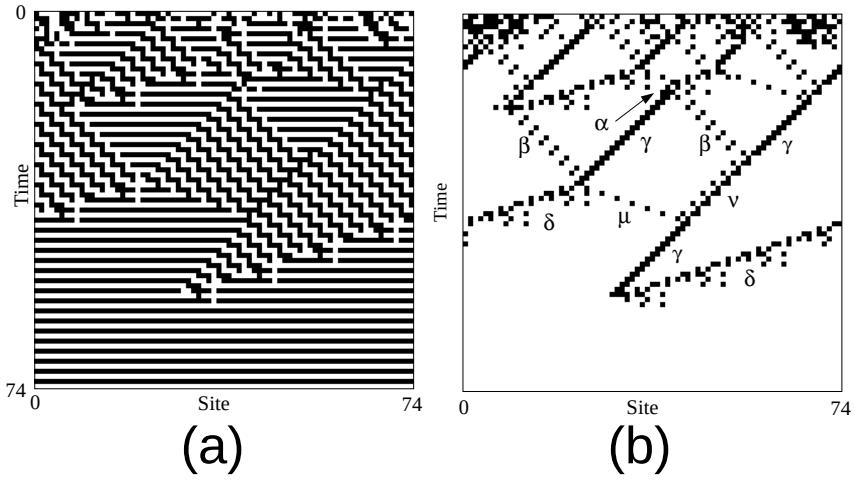

Figure 3.7: (a) Space-time diagram of ϕ_{sync5}, starting from a random IC. (b) Same space-time diagram as in (a) with the domains filtered out. After [DCMH95].

ϕ_{dens5} of figure 3.3, and ϕ_{sync1} to ϕ_{sync4} of figure 3.4. The resulting particle catalogs for these CAs are given in appendix A.

The emergent computational strategies of these evolved CAs can now be interpreted in terms of the domains and particles and their interactions, as summarized in the particle catalogs. ϕ_{dens2} in figure 3.3, for example, is a so-called *block expander*. This CA settles down to a black domain (1^*) very quickly on most ICs, except when there is a large enough block of consecutive 0s in the IC, in which case it starts expanding this block, until eventually the entire lattice becomes a white domain (0^*). The two particles on either side of the expanding white domain eventually meet up again because of the periodic boundary conditions. When they do collide, they annihilate each other, making the black domain disappear and the white domain take over. The "logic" behind this strategy is that statistically only on low density ICs is there a large enough chance to have a block of consecutive 0s in the IC (generally this block needs to be 7 or more cells wide). Thus, the presence of such a block is often a good indication that the overall density is low (i.e., $\rho_0 < 0.5$).

The strategy of ϕ_{dens2} is emergent in the sense that it makes use of higher level structures (domains and particles) that are embedded in its dynamical behav-

Particle Catalog								
Domains Λ			**Particles P**					
Label	Regular language		Label	Boundary	p	d	v
Λ^s	$\Lambda^s_0 = 0^*$, $\Lambda^s_1 = 1^*$		α	$\Lambda^s_0\Lambda^s_1, \Lambda^s_1\Lambda^s_0$	-	-	-
Λ^z	$\Lambda^z_0 = (0001)^*$, $\Lambda^z_1 = (1110)^*$		β	$\Lambda^z_0\Lambda^s_0, \Lambda^z_1\Lambda^s_1$	2	2	1
Interactions I			γ	$\Lambda^s_0\Lambda^z_1, \Lambda^s_1\Lambda^z_0$	2	-2	-1
Type	Interaction	Interaction	δ	$\Lambda^z_0\Lambda^s_1, \Lambda^z_1\Lambda^s_0$	4	-12	-3
decay	$\alpha \rightarrow \gamma + \beta$		μ	$\Lambda^s_0\Lambda^z_0, \Lambda^s_1\Lambda^z_1$	2	6	3
react	$\beta + \gamma \overset{0.84}{\rightarrow} \delta + \mu$	$\beta + \gamma \overset{0.16}{\rightarrow} \nu$	ν	$\Lambda^z_0\Lambda^z_1, \Lambda^z_1\Lambda^z_0$	2	-2	-1
	$\mu + \delta \rightarrow \gamma + \beta$	$\nu + \delta \rightarrow \beta$					
	$\mu + \nu \rightarrow \gamma$						
annihilate	$\mu + \beta \rightarrow \emptyset$	$\gamma + \delta \rightarrow \emptyset$					

Table 3.2: The particle catalog of ϕ_{sync5}.

ior. However, the "decision" whether the overall density of the IC is low or high is still made on a rather local basis, namely the presence or absence of a block of 0s. Therefore, ϕ_{dens2}'s fitness is only slightly higher than 0.5. ϕ_{dens5}, on the other hand, uses a much more sophisticated strategy where information is transferred across the lattice, and decisions are made simultaneously on different time and length scales.

The main idea behind ϕ_{dens5}'s strategy is shown in figure 3.8. In the space-time diagrams in this figure, initially there are two domains: a white domain (0^*) and a black domain (1^*). Where the white domain (on the left) meets the black domain (on the right), a checkerboard domain ($(01)^*$) is created with two particles traveling at equal but opposite velocities (-1 and +1, respectively), thus expanding this checkerboard domain. Where the black domain (on the left) meets the white domain (on the right), a zero-velocity particle is created that maintains the boundary between the two domains in the same place.

Now, when the white domain (initially) is smaller in length than the black domain, as is the case if figure 3.8(a), the particle with velocity -1 (the white-checkerboard boundary) will reach the zero-velocity particle before the other particle

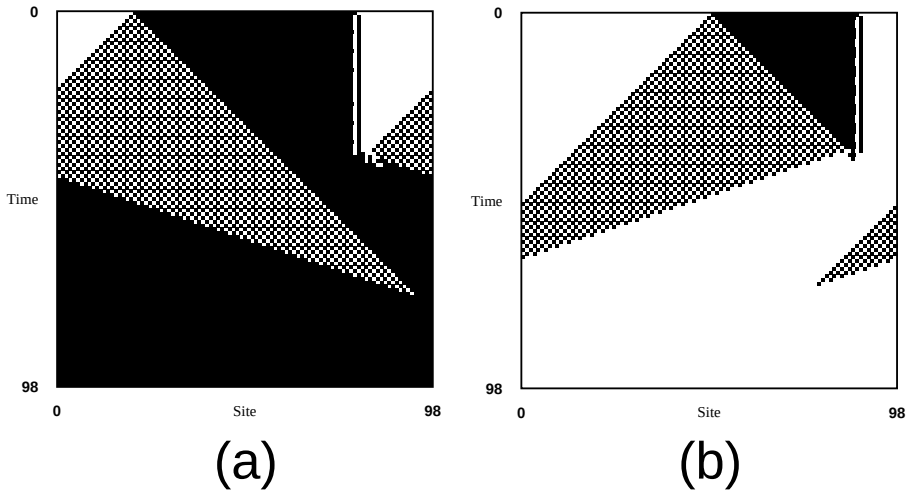

Figure 3.8: The general strategy of ϕ_{dens5}. (a) White domain smaller than black domain. (b) White domain larger than black domain.

(the checkerboard-black boundary) can, again because of periodic boundary conditions. The collision between the two particles results in the disappearance of the white domain and the appearance of another particle (the black-checkerboard boundary) that travels with a velocity of +3. Because of its higher velocity, this particle eventually "catches up" with the original particle with velocity +1 (the checkerboard-black boundary). Upon collision, these two particles annihilate, making the checkerboard domain disappear altogether. As a result, the black domain ends up occupying the entire lattice. Since the black domain was larger than the white domain to start with, the overall density of the IC was $\rho_0 > 0.5$, and the CA has made the correct classification.

A similar, but opposite, particle interaction happens when the white domain is larger than the black domain, as is shown in figure 3.8(b). In this case, the black domain disappears first, and after the checkerboard domain has eventually disappeared, the white domain occupies the entire lattice. Since in this case the overall density of the IC was $\rho_0 < 0.5$, the CA again made the correct classification.

Going back to a random initial configuration again, as shown in figure 3.3, ϕ_{dens5} quickly settles down into local regions of white, black, and checkerboard do-

mains after an initial transient period. From then on, the strategy just explained is used at different length and time scales to decide about local densities, until a final decision is made about the overall density of the IC. This way, the CA incorporates local information from different places into a final global decision. Consequently, this results in a much higher fitness than, for example, the more local block-expanding strategy of ϕ_{dens2}.

As can be seen in figure 3.3, the checkerboard domain is already present in ϕ_{dens2}. However, in this CA the checkerboard domain is not used in the strategy for classifying density (see e.g. [DMC94]). A few generations later, a CA (ϕ_{dens3}) with a very similar strategy to that of ϕ_{dens5} appears, which indeed uses the checkerboard domain which it "inherited" indirectly from ϕ_{dens2}. However, in this CA the black-white boundary is moving slowly to the right, resulting in a slight bias against white domains, since it decreases the size of white domains during the decision process. In the next CA (ϕ_{dens4}) this bias is fixed with a change in the particle's velocity from some small positive number to zero. Finally, ϕ_{dens5} is a "cleaned up" version of the basic strategy in the sense that some of the particles are simpler and that the transient time is shorter and the space-time behavior during this transient time is less "messy".

A similar structural analysis applies to the CAs shown in figure 3.4 for the global synchronization–1 task. These evolved CAs quickly settle down into locally synchronized regions with particles in between them. However, often these local regions are out of phase with each other. For example, at one particular time step one region is in the 0^* configuration, while the neighboring region is in the 1^* configuration. The particles in between these regions interact with each other, trying to resolve these phase differences. Over time, the computational strategies improve their ability to resolve these phase differences, resulting in increasing fitness values. Eventually, the GA finds CAs that make use of an additional domain (the zig-zag pattern) that serves similar purposes as the checkerboard domain in the density classification CAs.

So, as this brief review shows, by using the computational mechanics analysis the relevant information about the domains and particles and their interactions in evolved CAs can be extracted and collected in particle catalogs. As was mentioned in the previous chapter, in the CM framework itself no semantics is assigned to these discovered structures. However, in the case of the CAs that were evolved to perform a certain computational task, the particles can be interpreted as information carriers and particle interactions as the exchange and processing of information. In this sense, a CA's particle catalog can be viewed as a summary of the CA's *particle logic*, which underlies its computational strategy for performing a given task. Furthermore, the evolutionary history of these computational strategies can be described using this particle logic. For example, some of the differences in fitness between evolved CAs can be explained in terms of differences in particle velocities.

3.6 Results on the new tasks

As the previous section shows, the computational mechanics framework forms a useful tool for analyzing the computational strategies of the evolved CAs. However, one could ask how general the occurrence of domains and particles, and thus the applicability of the computational mechanics analysis, is in CAs that are evolved for performing computational tasks.

Using the same GA parameter settings as in the experiments reviewed above, several runs of the GA on the new tasks $\mathbf{T}_{sync2}$ and $\mathbf{T}_{sync3}$ were done. As with the density classification and original global synchronization tasks, on a small percentage of runs the GA did indeed find CAs with sophisticated, particle based, emergent strategies for performing the given tasks. Figure 3.9 shows space-time diagrams of typical CAs that occurred during such runs.

$\phi_{\text{sync-2a}}$ occurred at an intermediate stage during the evolution on the $\mathbf{T}_{sync2}$ task. Its fitness as measured during the GA run is 0.76. As figure 3.9 shows, $\phi_{\text{sync-2a}}$'s

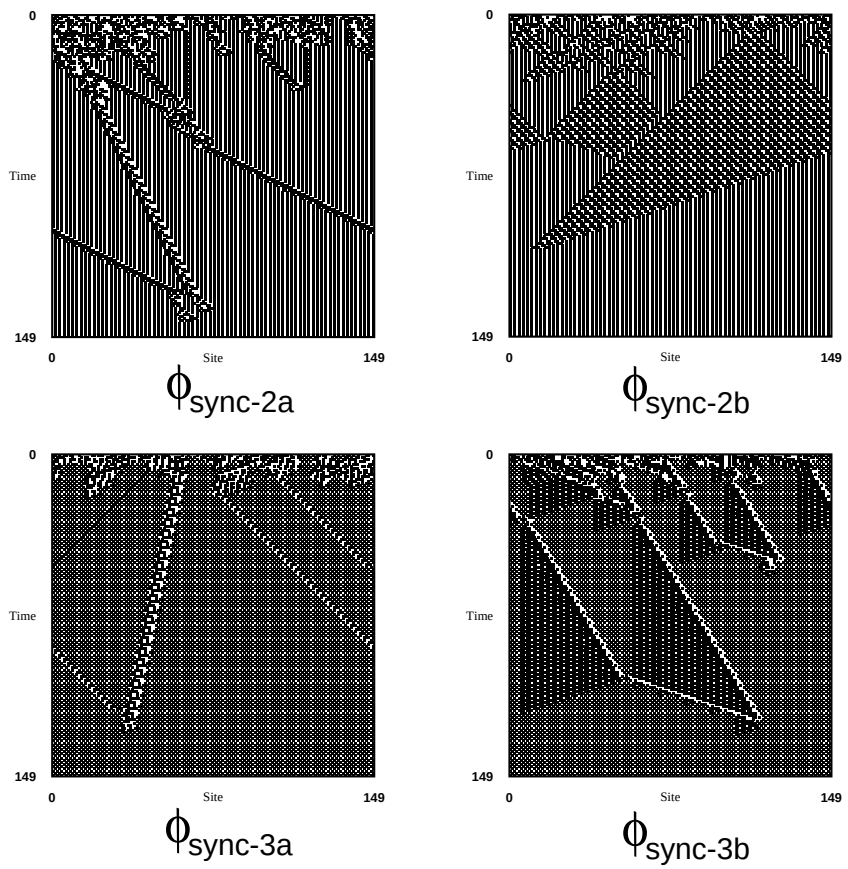

Figure 3.9: Typical results of evolving CAs for the $\mathbf{T}_{sync2}$ and $\mathbf{T}_{sync3}$ tasks. $\phi_{\text{sync-2a}}$ and $\phi_{\text{sync-3a}}$ are CAs that occurred early on in a GA run on the respective tasks. $\phi_{\text{sync-2b}}$ and $\phi_{\text{sync-3b}}$ are the best CAs found for these tasks.

dynamics quickly settle down, after some transient period, into local regions of vertical stripes (locally synchronized), with particles in between these regions. However, as in the CAs evolved for $\mathbf{T}_{sync1}$, these local regions can be out of phase with each other. For example, in one region the white vertical stripes are on the even cells, while in the neighboring region they are on the odd cells in the lattice. The particles in between the regions interact with each other, trying to resolve these phase differences.

$\phi_{\text{sync-3a}}$ is a similar CA that occurred at an intermediate stage during a GA run on $\mathbf{T}_{sync3}$. Its fitness is 0.74. $\phi_{\text{sync-3a}}$'s dynamics similarly settles down into regions that are locally synchronized (with the required checkerboard pattern), with particles in between these regions that try to resolve existing phase differences.

Both $\phi_{\text{sync-2a}}$ and $\phi_{\text{sync-3a}}$ are typical examples of CAs that appear once the GA has found non-zero-fitness CAs. In about half of the GA runs on the two new tasks, the GA never finds any CAs that have a fitness that is larger than zero (i.e., no solutions are found at all). However, in those runs where the maximum fitness in the population does get above this zero-fitness level, CAs similar in behavior to $\phi_{\text{sync-2a}}$ and $\phi_{\text{sync-3a}}$ appear quickly. In most of these runs, though, the GA is not able to find anything more sophisticated than these solutions, which are equivalent in behavior and fitness to, for example, ϕ_{sync3} in figure 3.4.

On a small percentage of runs, however, the GA did find high-fitness solutions. $\phi_{\text{sync-2b}}$ and $\phi_{\text{sync-3b}}$ are the best solutions found (over all GA runs) for the $\mathbf{T}_{sync2}$ and $\mathbf{T}_{sync3}$ tasks, respectively. The fitness of both these CAs is 1.00. As with the original global synchronization task, the GA was able to find CAs that use an additional domain, with corresponding particles, to resolve phase differences between locally synchronized regions, as can be seen in the space-time diagrams in figure 3.9.

In fact, the particle logic of $\phi_{\text{sync-2b}}$ is exactly the same as that of ϕ_{sync5}, except for a slight difference in the probabilities of interaction results for the one interaction with multiple results. For a comparison, see ϕ_{sync5}'s particle catalog in table 3.2 and $\phi_{\text{sync-2b}}$'s particle catalog in appendix A. Figure 3.10 shows space-time diagrams of both CAs, illustrating the similarity in their computational strategies.

Given this similarity, one could ask whether this particle logic is "perfect" for performing a task similar to global synchronization. Unfortunately, the answer is negative, at least for arbitrary lattice sizes. Note that because of the periodic boundary conditions, on certain lattice sizes it is possible to have an infinite sequence of $\beta + \gamma \rightarrow \delta + \mu$ and $\mu + \delta \rightarrow \beta + \gamma$ interactions, without ever settling down to the globally synchronized state. Figure 3.11 shows an example of such a sequence of interactions in a space-time diagram of $\phi_{\text{sync-2b}}$.

So, the particle logic of ϕ_{sync5} and $\phi_{\text{sync-2b}}$ is not completely perfect. However, as the fitness of 1.00 for both CAs shows, the globally synchronized state is reached

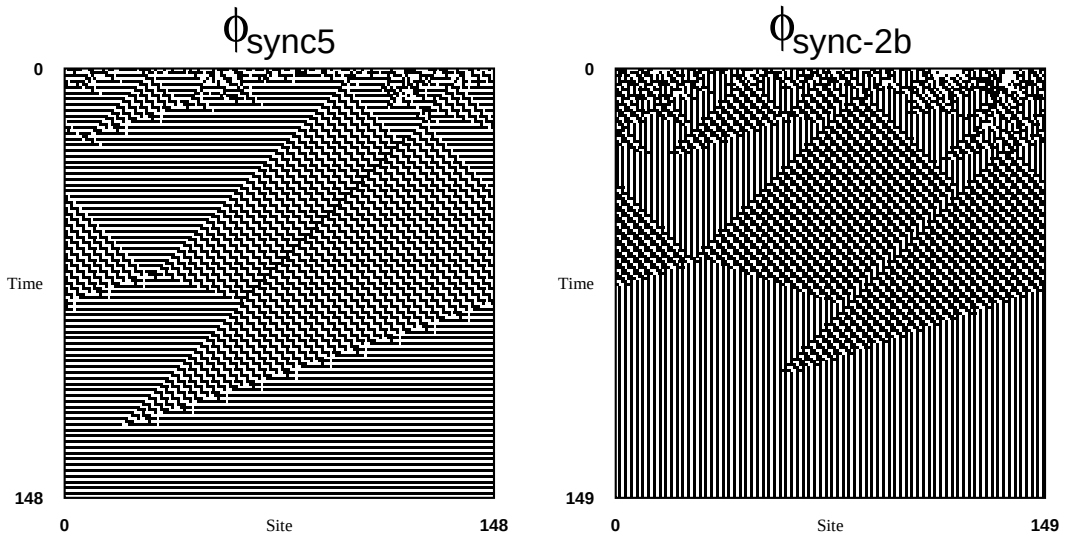

Figure 3.10: Space-time diagrams of ϕ_{sync5} and $\phi_{\text{sync-2b}}$, illustrating the similarity in their computational strategies.

every time on a random sample of ICs. Furthermore, it is not the only particle logic that can perform the global synchronization task (or some variant of it). For example, $\phi_{\text{sync-3b}}$'s particle logic is rather different, as the space-time diagram in figure 3.9 and the particle catalog in appendix A show. $\phi_{\text{sync-3b}}$'s fitness is also 1.00, however.

Concluding, on these new tasks too, the highly fit CAs that are found by the GA make use of domains and particles to perform the given task. Sometimes the same particle logic can be used to perform the related tasks, as the similarity between the particle logics of ϕ_{sync5} and $\phi_{\text{sync-2b}}$ shows. But, high performance is not restricted to this one particular strategy, as the particle logic of $\phi_{\text{sync-3b}}$ shows. However, all these CAs do make use of domains and particles, and the computational mechanics framework can be applied directly to analyze their emergent behavior. The particle catalogs of the four CAs shown in figure 3.9 are presented in appendix A.

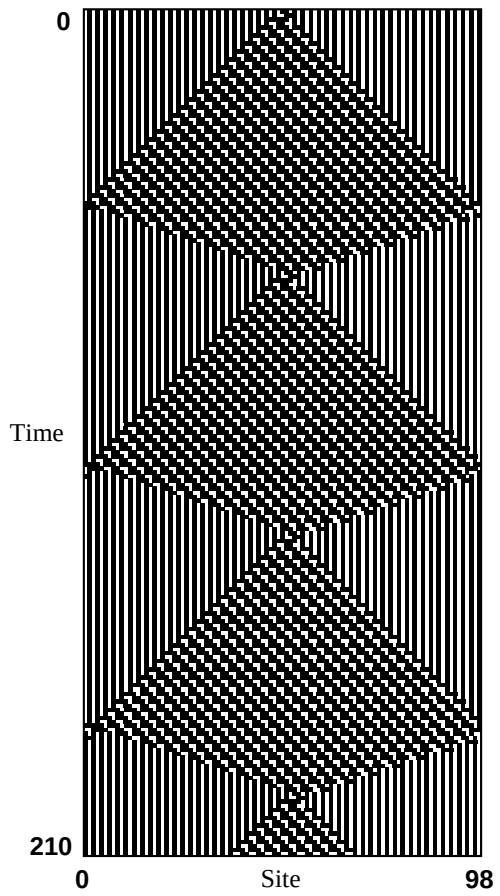

Figure 3.11: An example of a space-time diagram for $\phi_{\text{sync-2b}}$ where the CA never settles down to global synchronization.

3.7 Related work

Besides the work described in this chapter, related work on evolving cellular automata has been done by other researchers. In a variation on the work described above, Sipper and colleagues used a GA to evolve non-uniform CAs, i.e., CAs where each cell has its own lookup table. They used a co-evolutionary scheme, where the individuals in the population are the different lookup tables of the cells in the lattice [Sip97]. In addition to the density classification and global synchronization tasks, Sipper and colleagues used GAs to evolve CA for other tasks such as ordering, rectangle filling, and random number generation. Some of these tasks were done on 2-dimensional

lattices.

Andre and colleagues used genetic programming (GP) to evolve cellular automata for the density classification task [ABIK96]. Using a population of size 51,200 divided into 64 sub-populations of size 800, their GP algorithm was able to find a CA that was slightly better than the best known CA for density classification at that time.

Paredis used a version of the genetic algorithm where the CAs and the ICs are co-evolved and reports on some impediments that this creates for finding good solutions [Par97]. In particular, large fluctuations appear when the best CA fitness in the population is plotted over time. These fluctuations are the result of an "arms race" typically observed in predator-prey systems, preventing the GA from ever finding good overall solutions.

However, Juillé and Pollack came up with a way to avoid these impediments in a co-evolutionary GA and report on their success in co-evolving CAs and ICs for the density classification task [JP98a, JP98b]. Together with the co-evolutionary scheme, they used a technique called *resource sharing*, where a CA gets extra credit when it correctly classifies an IC that is misclassified by most other CAs in the population. Not only did they find a CA that has even higher performance than the one found by Andre and colleagues (using GP), but their co-evolutionary GA also found good solutions more often than a regular GA. Juillé and Pollack claimed that these improvements were mainly due to the co-evolution rather than the resource sharing. However, subsequent experiments by others provide evidence that the improvements observed when both techniques are used together depend largely on resource sharing alone [WMC99].

As mentioned earlier, it has been shown that no two-state finite radius CA can perform the density classification task perfectly [LB95, Das98]. However, Fukś has shown that using a combination of two elementary CAs (rule 184 and 232), this task can be performed perfectly [Fuk97]. The main idea here is to first iterate ECA 184

for t_1 time steps and subsequently iterate ECA 232 for another t_2 time steps. This way, this pair of CAs will correctly classify every possible IC.

Most of the related work described in this section has mainly focused on designing alternative evolutionary search algorithms and on finding better solutions for performing the density classification task. However, these studies have not, at least not directly, addressed the question about the relation between dynamics and emergent computation in (evolved) CAs. Consequently, the research summarized in this section does not provide more insight into this relation, and thus does not contribute directly to the main goal of this dissertation.

Chapter 4

Particle Models of Emergent Computation

The work described in the previous chapter provides insight into the relations among dynamics, emergent computation, and evolution in cellular automata. However, as argued in the introduction, what is still missing is a more quantitative and predictive analysis of these relations. The class of particle models introduced in this chapter provides a tool for such an analysis. With this new class of models, the dynamics of evolved CAs is modeled at the emergent level of domains, particles, and particle interactions. In particular, using a CA's particle logic as summarized in its particle catalog, and some additional information about the frequency of particle occurrences, the CA's computational strategy is simulated directly, abstracting away from the underlying cells in the CA lattice.

This class of particle models is sufficiently specific that it can be used to make quantitative predictions of an evolved CA's computational performance on a given task. Furthermore, the models can be used to predict how changes in an evolved CA's computational strategy lead to changes in its performance. These changes can then be related to the evolutionary history of the evolved CAs. This is done by using the particle models to determine quantitatively the extent to which differences in the computational strategies of evolutionarily related CAs contribute to differences in these CAs' performances. Furthermore, these results show how modifications in the CAs' computational strategies made during the evolution led to improved fitness values. This way, both the relation between dynamics and emergent computation (i.e., how does the CA dynamics give rise to emergent computation) and the evolutionary history of emergent computation in CAs can be studied more formally.

Before the particle models are introduced, however, a number of other concepts need to be discussed. First, the notion of *condensation time* is introduced. As in any modeling situation, the particle models incorporate some simplifying assumptions. These assumptions are listed next. After that, the class of models is introduced in full detail. A model is not useful, though, if it does not provide a more concise or efficient description than the complete description of a system's behavior itself. To show that

the class of particle models indeed forms a more concise and efficient description of the (global) dynamics of the evolved CAs, the computational complexities of CAs and their corresponding particle models are compared.

4.1 The condensation time

As the space-time diagrams in the previous chapter show, most evolved CAs quickly settle down, after some initial transient phase, to spatial configurations consisting of regular domains and particles. Note that this notion of transient phase is slightly different from its usual meaning, which is the phase before the entire CA lattice settles down to some fixed point or periodic configuration. In this section, the notion of transient phase as used here is defined formally.

First, recall that when a CA's domain transducer is used to filter a lattice configuration, it outputs a "flag", or sequence of non-domain symbols, each time the regular pattern of a domain is violated. These violations can either be proper particles or more or less "random" local configurations that are not part of a domain or a particle. When such a violation is indeed a particle, however, the corresponding sequence of non-domain symbols uniquely identifies the type and phase of this particle, as discussed in section 2.4.3.

So, when all the particles that can occur in a CA's dynamics are known, the *signatures* (i.e., the unique sequences of non-domain symbols) that identify their respective types and phases can be collected. Taking this set of signatures of all particles and their phases, the domain transducer can be augmented to recognize, in addition to the domains, all the known particles. This augmented *domain-particle transducer* can then again be used to filter CA lattice configurations. This transducer now outputs a "flag" (indicating a violation) only when some spatial configuration is read that does not correspond to either a domain or a particle. A detailed example of how to construct such an augmented transducer can be found in [HC97].

The *condensation time* t_c in a CA's space-time diagram is now defined as the smallest time step t for which the domain-particle transducer, when scanning the entire lattice configuration $\mathbf{s}_t$, does *not* output any flags. In other words, the condensation time is the *first* time step at which the CA lattice consists entirely of proper domains and particles. The phase up to the condensation time is called the *condensation phase*, a period during which there still are some unrecognized (in terms of domains and particles) or "random" spatial configurations present. At the condensation time, however, the lattice has *condensed* into a lattice configuration that consists solely of domains and particles. Thus, the condensation phase is what is used as the definition of transient phase here. Figure 4.1 illustrates this notion of condensation time in a space-time diagram of ϕ_{dens5}.

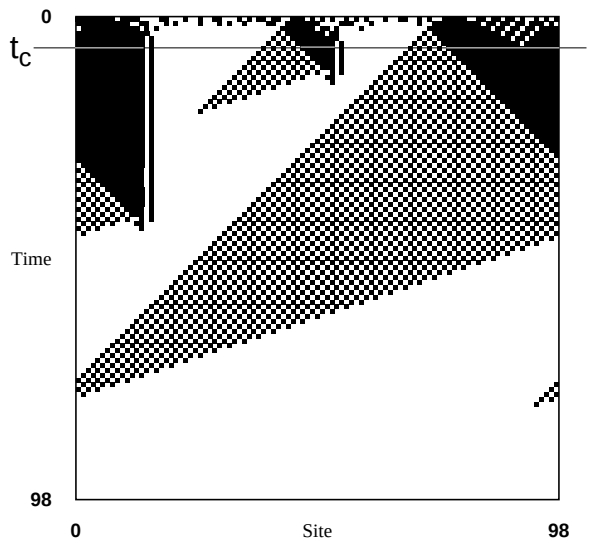

Figure 4.1: Example of the condensation time, marked by the horizontal line labeled t_c. During the condensation phase, there are still some non-domain/particle configurations present in the lattice.

The exact value of the condensation time t_c obviously depends on the lattice size and on the initial configuration with which the CA starts. On some ICs a CA will condense into domains and particles more quickly than on others. An average condensation time $\overline{t_c}$ can be calculated by averaging the t_c values over a large set

of random ICs. For example, in the particular space-time diagram shown in figure 4.1 (with lattice size $N = 99$), $t_c = 6$. The average condensation time for ϕ_{dens5}, measured over 10,000 random ICs on a lattice of size $N = 149$, is $\overline{t_c} = 16$, with a standard deviation of 5.8.

In chapter 6 it is proved that for one particular CA the condensation time is bounded by a constant (in particular, $t_c \leq 3$ for that CA). Furthermore, in that chapter it is shown how t_c scales with the lattice size and how the average t_c can be calculated directly as a function of the lattice size.

4.2 Simplifying assumptions for the particle models

The class of particle models incorporates a number of assumptions that will simplify the simulation of an evolved CA's computational strategy. This section lists and explains these simplifying assumptions, which are all related to particles and particle interactions, since these form the main components of the particle models.

4.2.1 Particle probability distribution at t_c

As a first simplifying assumption in the model class, the detailed dynamics of a CA during the condensation phase is ignored. It is assumed that the sole purpose of the condensation phase is to create a particular configuration of domains and particles at the condensation time t_c. Since particles are partly defined by the domains between which they form a boundary, stating the types and locations of the particles (i.e., the *particle configuration*) at t_c is sufficient to express the entire lattice configuration at t_c. The types and locations of the domains are implicitly taken into account in such a particle configuration.

As with the actual value of t_c itself, the particular particle configuration at t_c depends on the IC. However, the condensation phase dynamics induces a probability

distribution of particle occurrences at t_c. More formally, during the condensation phase the CA maps the probability distribution $\Pr[\mathbf{s}_0]$ of ICs to a probability distribution $\Pr[\mathbf{s}_{t_c}]$ of lattice configurations at t_c which consist of proper particle configurations. In other words, measured over a large set of random ICs, there is a certain probability distribution of how often and where particles occur at t_c. In the particle models, the condensation phase dynamics is replaced by an approximation of this *particle probability distribution* (PPD) at t_c. This approximation of the PPD at t_c can then be used to generate particle configurations at t_c for the purpose of the particle models (see below).

This assumption splits the CA dynamics in two distinct phases: (1) the condensation phase and (2) the particle phase. Consequently, these phases are modeled differently in a CA's particle model (as explained below).

4.2.2 Zero-width particles

In the particle models, particles are considered to have no width. Obviously, this is not the case in actual CAs, where particles are usually several cells wide. Figure 4.2, for example, shows a particular space-time diagram of ϕ_{sync3}. The particles in this space-time diagram range roughly from 4 to 10 cells wide. However, for the sake of simulating a CA's computational strategy, these particles are treated as points in space.

Furthermore, the result of a particle interaction is not always exactly at the same location as where the interacting particles originally collided. For example, in the interaction labeled "A" in figure 4.2, the particle that results from this interaction is shifted to the left compared to where the three interacting particles originally collided. In the particle models, because of the assumption of zero-width particles, an interaction result is assumed to appear at the same location as where the interacting particles collide.

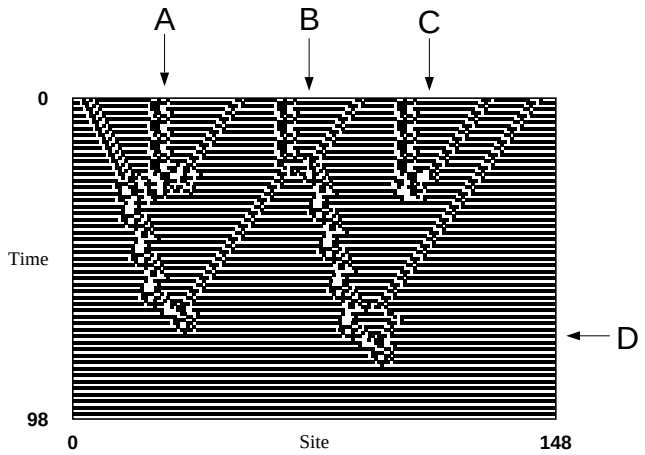

Figure 4.2: A space-time diagram of ϕ_{sync3} illustrating the simplifying assumptions of the particle models. The labels A, B, C, and D refer to the particle interactions.

4.2.3 Two-particle interactions only

In the particle models only interactions between *pairs* of particles are considered. In the actual CAs three or more particles can interact, as is shown in figure 4.2. In the particle interaction labeled "A", for example, three particles collide, and their interaction creates one new particle. In the particle models, however, particles are treated as dimensionless points in space, so it is highly unlikely that three or more particles will meet at exactly the same position at the exact same time. Because of this, the particle catalogs of the CAs analyzed here show interactions only between pairs of particles, even though in the actual CAs interactions among three or more particles can occur.

Just as the number of incoming particles is limited to two, the number of outgoing particles from an interaction is also limited to two. This might seem an unreasonable restriction at first. But, in fact, in all the CAs that were evolved and analyzed, no two-particle interactions have been observed that produce more than two particles as an interaction result. Thus, this limitation is empirically justified. Furthermore, this simplifying assumption can be easily relaxed, if needed.

4.2.4 Instantaneous interactions

In an actual CA a particle interaction can temporarily lead to non-domain/particle configurations. For example, in interaction "A" in figure 4.2, there are a few time steps during the interaction where locally the lattice cannot be described in terms of domains and particles. Soon after that, however, an interaction result in the form of a proper particle appears. Similarly, in the interaction labeled "D" it takes several time steps after the time of collision before the interacting particles have annihilated. In the particle models this "delay" is ignored and interactions are considered instantaneous. In other words, when two particles collide they are replaced immediately with their interaction result.

4.2.5 Stochastic approximation of phase dependencies

As mentioned in section 2.4.4, sometimes the result of a particle interaction depends on the respective phases that the interacting particles are in at the time of collision. For example, both the interactions labeled "B" and "C" in figure 4.2 involve the same particle types. However, the interactions have different outcomes. In interaction "B" two new particles are created, while in interaction "C" the interacting particles annihilate each other. This is solely a result of the different phases that each of the particles are in at the time of collision.

When a pair of particles in a CA has more than one possible interaction result, it can be empirically estimated (over a large set of ICs) what fraction of these interactions lead to each of the possible results. These fractions can then be used as probabilities of interaction result occurrences in the CA's particle catalog. For example, in ϕ_{sync5}'s particle catalog (table 3.2) there is one interaction that can have two possible outcomes: $\beta + \gamma \rightarrow \delta + \mu$ and $\beta + \gamma \rightarrow \nu$. The first interaction occurs with (empirically measured) probability 0.84 and the second one with probability 0.16.

The particle models do not keep track of the phases of the individual particles. However, in the case of multiple possible interaction results, this phase-dependence

is approximated by a stochastic choice of interaction result using the probabilities given in the particle catalog. For example, when during a simulation of ϕ_{sync5}'s computational strategy a collision between a β particle and a γ particle occurs, a choice between the interaction results $\delta + \mu$ and ν is made according to the estimated probabilities.

4.2.6 Summary

In summary, the simplifying assumptions incorporated in the particle models are:

1. The CA dynamics during the condensation phase is replaced with an approximation of the PPD at t_c.

2. Particles are assumed to have zero width, and spatial shifts in interaction results are ignored.

3. Only interactions between pairs of particles are considered, with at most two particles as interaction result.

4. Particle interactions are assumed to be instantaneous.

5. The phase dependencies in particle interactions are approximated by a stochastic choice of interaction result (based on empirically measured probabilities).

With the notion of condensation time introduced and the simplifying assumptions listed, the class of particle models can now be introduced in detail.

4.3 The class of particle models

The main idea behind the class of particle models is to use a CA's particle catalog, together with an approximation of the PPD at t_c, to simulate the CA's computational strategy directly. In other words, the CA's global dynamics is modeled at the level of

the emergent structures—formalized as domains, particles, and particle interactions—using the CA's particle logic. It is a *class* of models, since it forms a general framework for modeling and analyzing the dynamics of CAs, but each CA needs it own model instantiation. A preliminary version of this class of particle models was introduced earlier in [HCM98]. Here, the class of models is introduced in full detail.

To model the (emergent) dynamics of a CA, first the particle catalog of the CA needs to be constructed. This catalog contains information about the domains, particles, and particle interactions that occur in the CA, and it forms a concise description of the CA's particle logic. Next, an approximation of the PPD at t_c is needed. It turns out that, in general, finding a concise and accurate approximation of the PPD at t_c is a difficult problem. This problem, the reason why it is generally difficult, and some CA-specific solutions are discussed in section 6.2. Accurate but less concise approximations are easily obtained (see below).

With the CA's particle catalog and an approximation of the PPD at t_c at hand, the CA's computational strategy can now be simulated directly with the following algorithm.

1. Get an initial particle configuration at t_c from the approximation of the PPD at t_c. Set $t = t_c$.

2. If there are two or more particles in the lattice, go to step 3. Otherwise, go to step 6.

3. Calculate the next time step t_i at which two particles will collide. If $t_i > M$ (the maximum number of time steps allowed), go to step 6.

4. Set $t = t_i$. Replace the colliding particles with their interaction result. Update the positions of the other particles in the lattice.

5. Go to step 2.

6. Set $t = M$. Update the positions of the remaining particles in the lattice. Stop.

The implementation of step 1 depends on the particular approximation of the PPD at t_c that is used. Here, the following approximation, which is extremely accurate, is used. Starting with a random IC, the CA being modeled is run up to the condensation time t_c. Recall that t_c is determined by using the domain-particle transducer to scan the CA lattice configurations until the first time step at which the entire lattice consists purely of domains and particles. While the domain-particle transducer is scanning the lattice, it also keeps track of the types and locations of the particles it encounters. Thus, when t_c is reached, the particle configuration at that time step is readily available. This particle configuration, copied from the actual CA, can then be used as the initial particle configuration at t_c in the CA's model. This way, the actual value of t_c can be used in the CA's model. Every time a CA's particle model is run, a new particle configuration at t_c is thus created by running the CA up to t_c on a new random IC.

In step 2, a check for the total number of particles in the lattice is done. In case of 0 or 1 particles, there can never be any particle interactions. In that case, the intermediate steps of the algorithm are skipped and the final step is executed immediately. Otherwise, the algorithm proceeds with the next step.

Step 3 of the algorithm is straightforward. It is now known where the particles are in the lattice and of what type they are. Furthermore, the velocities of the different particle types are listed in the particle catalog. So, for each pair of neighboring particles in the lattice it can be calculated, with a simple algebraic formula, when these particles would collide if they were the only pair of particles in the lattice. For example, if there is one particle with velocity v_1 at site s_1 and one particle with

velocity v_2 at site s_2, then their interaction time t_i is simply calculated as:

$$s_1 + v_1 t_i = s_2 + v_2 t_i \Rightarrow (v_1 - v_2)t_i = s_2 - s_1 \Rightarrow t_i = \frac{s_2 - s_1}{v_1 - v_2}$$

In some cases, for example when the particles are actually moving away from each other, this formula needs to be adjusted slightly to take the periodic boundary conditions of the lattice into account. Taking the smallest one of the collision times of all the neighboring particle pairs gives the desired t_i. Thus, if there are n particles in the lattice, this step of the algorithm requires n simple calculations, one for each particle and its neighbor to, say, its right side.

Note that the value found for t_i in step 3 can be larger than the maximum number M of time steps allowed. This can either be because the next collision happens at a time step larger than M or because the only remaining particles in the lattice all travel with the exact same velocity, in which case $t_i = \infty$. When $t_i > M$, the following two steps are skipped and the algorithm jumps directly to the final step.

In step 4 the particle interactions listed in the particle catalog need to be consulted. Knowing what the two particle types are that collide at t_i, the interaction result can be found in the particle catalog. This might involve choosing, at random, one of multiple possible interaction results, as explained in the simplifying assumptions above. The two interacting particles are then replaced with their interaction result, which might be an annihilation, in which case there are no new particles coming out of the interaction. The interaction is considered to be instantaneous, as explained in the simplifying assumptions above. Lastly, the positions of the other particles in the lattice are updated. Again, this is a straightforward calculation for each particle: if the current position at time t of a particle is s, then its new position at time step t_i is $(s + v(t_i - t)) \bmod N$, where v is the particle's velocity and N is the lattice size.

Step 5 completes the main loop of the algorithm, jumping back to step 2 where the total number of particles is checked again, since this number might have decreased after the particle interaction has been completed.

Step 6, the final step of the algorithm, is reached only when there are no more particle interactions between the current time step and time step M. In this final step, the positions of the remaining particles, if any, are updated to reflect their final positions at time step M. Then the algorithm is terminated. Note that if it so happens that all particles have annihilated each other before time step M, it is implicitly known which domain is left occupying the entire lattice, since the information about the domains in the lattice is implicitly present in the particle configurations at each time step, as explained in section 4.2.1. So, when the last two particles in the lattice annihilate each other, it is known which domain occupies the entire lattice.

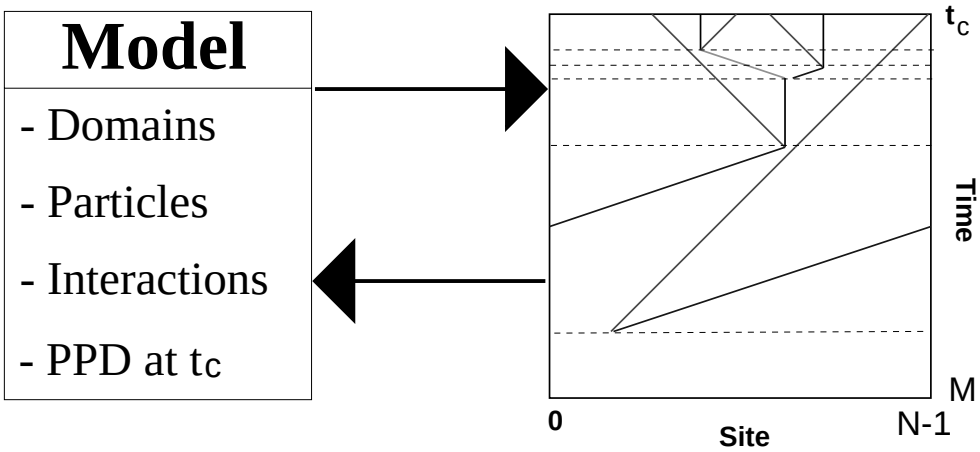

Figure 4.3: A schematic overview of a CA's particle model.

Figure 4.3 shows schematically how a CA's particle model works. The model contains information about the domains, particles, particle interactions, and an approximation of the PPD at t_c. Using this information, first a random initial particle configuration at t_c is created. Then, the next interaction time t_i is calculated, indicated by the first dotted line in the space-time diagram. The two colliding particles are then replaced with their interaction result, which is found in the particle catalog. Finally, the positions of the other particles in the lattice are updated. This cycle of calculating interaction times, replacing colliding particles with their interaction result, and updating the positions of the other particles, is repeated until, in this

example, all particles have annihilated each other. The subsequent interaction times are indicated by the dotted lines in the space-time diagram. The arrows indicate the interplay that occurs at these interaction times between the static information in the model's particle catalog and the dynamic information in the model's space-time diagram.

4.4 Predicting the performance of evolved cellular automata

As just mentioned, if all particles have annihilated each other by time step M, it is implicitly known which domain is left occupying the entire lattice. In other words, it is know whether a CA's particle model has settled down to the correct answer state in the context of a given computational task. So, similar to calculating the actual performance of a CA on a computational task, the *model-predicted* performance of a CA can be calculated with the CA's particle model. Instead of starting with a random set of ICs, a random set of initial particle configurations at t_c is used, generated by the approximation of the PPD at t_c. The CA's computational strategy is then simulated, using the information in the CA's particle catalog, by running the above algorithm on each of the initial particle configurations.

The fraction of initial particle configurations on which these model simulations settle down to the correct final domain (i.e., correct answer state) within M time steps, can be taken as a performance prediction for the CA. Furthermore, a CA's particle model can be used to predict changes in performance given changes in the CA's computational strategy. For example, by changing the velocity of a certain particle in the CA's particle catalog, and running the algorithm again on the same set of initial particle configurations, the CA's particle model can be used to predict what influence this change in particle velocity will have on the CA's performance.

The class of particle models provides a tool for a formal study of the relation

between dynamics and emergent computation in evolved CAs. First of all, the models give a concise description of the emergent dynamics of CAs. In particular, the models provide a formal description of the notion of a computational strategy in CAs. This description includes an algorithmic procedure with which these computational strategies can be simulated directly. Furthermore, the models can be used to relate quantitatively the dynamics of a CA (in terms of domains, particles, and particle interactions) to its computational performance on a given task. In other words, given a CA's (emergent) dynamics, its performance can be predicted using the particle models. So, the models show how patterns in a CA's dynamics give rise to emergent computation. Finally, by using the models to study CAs that were evolved by a GA, they can show how changes that occurred in the CAs' computational strategies during the evolution gave rise to the observed increases in fitness over time.

The class of models, including programs for collecting the necessary information to construct a CA's particle catalog, is fully implemented. Currently, this collection of programs consists of $\pm$20,000 lines of `C++` code and includes a graphical user interface. The implementation of the model reads a CA's particle catalog and an approximation of the PPD at t_c, and can then be used to calculate performance predictions for this CA or to generate space-time diagrams of the CA's particle logic. In fact, the space-time diagram in figure 4.3 was generated with this program, using the particle catalog of ϕ_{dens5}.

4.5 The computational complexity of cellular automata and their particle models

Generally, a model is expected to be a more concise description of a system and its behavior than the complete description of the system itself. Otherwise, there is typically little gain in using a model. A model can be more concise than the system it models because it is simpler to describe it, or because it is faster to run it, or perhaps

both.

One way to show that a CA's particle model is indeed more concise than a CA itself, is by comparing the computational complexities of a CA and its model. The *computational complexity* of a problem or an algorithm is the amount of resources (e.g., space or time) that are needed to solve the problem or to run the algorithm [HU79, Mor98]. In this section, both the space complexity and the time complexity (using both serial and parallel models of computation) of a CA are compared to those of its particle model. When relevant, worst case, average case, and best case complexities are considered.

4.5.1 The computational complexity of cellular automata

Space complexity

To calculate the space complexity of a k-state, radius r CA, it is necessary to know how much storage space, or memory, is needed to run the CA. Generally, two items need to be stored to run a CA: (1) the CA update rule ϕ and (2) the CA lattice configurations $\mathbf{s}_t$.

The update rule ϕ is usually represented as a lookup table with k^{2r+1} entries (see section 2.1). Thus, this representation requires $\mathcal{O}(k^r)$ memory space. For the lattice configurations $\mathbf{s}_t$, only two arrays of length N, where N is the lattice size, are necessary. One array is used for the current lattice configuration $\mathbf{s}_t$, and the second array is used to construct the configuration $\mathbf{s}_{t+1}$ at the next time step, after which the two arrays are swapped. The two arrays require $\mathcal{O}(N)$ memory space, independent of the number of time steps for which the CA is run.

One could argue that a counter for keeping track of the time steps is also necessary, especially if one wants to run the CA for a certain number of times steps, $M = 2N$ in this case. This requires $\mathcal{O}(\log M) = \mathcal{O}(\log N)$ memory.

Thus, the total memory requirement for running a CA is $\mathcal{O}(k^r + N + \log N) = \mathcal{O}(k^r + N)$. This puts a CA in the complexity class ExpSpace. When the radius r

is considered fixed, as in most of the evolving cellular automata experiments, where $r = 3$ is used, a CA ends up in the complexity class PSPACE.

The above complexity analysis is for CAs in general, using an explicit lookup table representation. In some cases, however, the space complexity of a CA can be reduced. For example, instead of using an explicit lookup table, the update rule ϕ can be represented as a Boolean function of the current states of the cells in a local neighborhood μ. See for example table 1 in the appendix in [Wol94] for a list of the shortest possible Boolean expressions for all the 256 elementary CAs. Such a representation generally requires an amount of memory that is polynomial in r, as compared to $\mathcal{O}(k^r)$ for an explicit lookup table. However, finding the shortest representation for an arbitrary boolean function is an NP-complete problem (see e.g. [GJ79]), making it impractical in general.

Similarly, for some CAs, in particular the "additive" ones, the update rule ϕ can be written as an algebraic expression [MOW84]. *Additive* CAs are ones for which the next state value s^i_{t+1} can be written as a linear sum of the current state values in cell i's local neighborhood μ^i. This representation also requires an amount of memory that is polynomial (in particular, linear) in r. However, a concise algebraic representation of ϕ is possible only for a limited set of CAs, again making it impractical in general.

Time complexity

To analyze a CA's time complexity, first a serial model of computation is assumed, such as a Turing machine. Furthermore, it is assumed that the CA is run for $M = 2N$ time steps, as in the EvCA experiments; see section 3.3.2.

Updating one cell in the CA lattice is a constant-time operation. An update consists of reading the local neighborhood configuration η^i of a cell i, followed by a table entry lookup to get the new state of cell i. The local neighborhood configuration η^i can be interpreted as a number written in base k, which forms the index of the entry in the lookup table that contains the new state of cell i. Thus, a one-cell update

requires time $\mathcal{O}(1)$. One complete update of an N-cell lattice then takes time $\mathcal{O}(N)$. When the CA is run for $M = 2N = \mathcal{O}(N)$ time steps, it follows directly that the time complexity of running a CA is $\mathcal{O}(N^2)$, i.e., a CA is in the polynomial time complexity class P.

Next, a parallel model of computation is assumed where a number of processors that is polynomial in the input size is allowed. Since the input size here is the lattice size N, the number of processors that is allowed is $\mathcal{O}(N^{\mathcal{O}(1)})$.

The state s^i_{t+1} of a cell i at time $t+1$ cannot be known until the states of the cells in the local neighborhood η^i are known at time t. In other words, the lattice configurations $\mathbf{s}_t$ have to be constructed explicitly for every time step t, and the number of global update steps cannot be decreased by using more than one processor. However, using N processors in parallel, which is linear in N, one global update can be done in constant time, i.e., in time $\mathcal{O}(1)$. Thus, running a CA for $M = 2N = \mathcal{O}(N)$ time steps on a parallel computer will take time $\mathcal{O}(N)$. So, even for a parallel model of computation, the time complexity of a CA is still in P, but is reduced from quadratic to linear.

Note that it is not surprising that after moving from a serial model to a parallel model of computation the time complexity of a CA is still in the polynomial class P. It has been proved that predicting the state s^i_t of a cell i at a certain time t is a P-complete problem (see e.g. [GHR95]). This implies that it is unlikely that running a CA can be efficiently parallelized. It appears to be an "inherently sequential" process.

For some special CAs, however, predicting s^i_t can be done in $\mathcal{O}(\log t)$ or $\mathcal{O}(\log^2 t)$ time, placing them in the complexity class NC of efficiently parallelizable problems. Examples of such parallelizable CAs are additive CAs and so-called quasi-linear CAs [Moo97, Moo98]. In general, however, CA prediction is P-complete and, unless NC=P[1], it cannot be done in less than linear time, even on a parallel computer.

[1]It is not known whether NC=P, just as it is not known whether P=NP.

4.5.2 The computational complexity of the particle models

Space complexity

As with CAs, the space complexity of a particle model is the amount of memory needed to run that model. Generally, three items need to be stored to run a particle model: (1) the approximation of the PPD at t_c, (2) the particle catalog, and (3) the current particle configuration.

The amount of memory needed for the approximation of the PPD at t_c depends on the approximation that is used. In the method used here, the CA being modeled is run up to t_c, which consequently requires the amount of memory necessary for running the CA, which is of the order $\mathcal{O}(k^r + N)$, as calculated above. However, in chapter 6 it is shown that more concise approximations are possible. So the above space requirement can be considered an upper bound.

A particle catalog consists of several items: (1) a list of domains $\mathbf{\Lambda}$, (2) a list of particles $\mathbf{P}$, and (3) a list of particle interactions $\mathbf{I}$. A constant amount of memory is needed to store a single domain, since all that is strictly needed is a label that uniquely identifies the domain. So, the space requirement for storing the list of domains is proportional to the number of domains, i.e., $\mathcal{O}(|\mathbf{\Lambda}|)$.

In general, given $|\mathbf{\Lambda}|$ domains, the number of particles is $|\mathbf{P}| = \mathcal{O}(|\mathbf{\Lambda}|^2)$, since there has to be at least one particle (i.e., boundary) for each combination of two domains. Each particle in the particle catalog requires a constant amount of memory for storing the particle's label, type of boundary, periodicity, displacement, and velocity. Thus, the space requirement for the list of particles is $\mathcal{O}(|\mathbf{\Lambda}|^2)$.

Given $|\mathbf{P}|$ particles, the number of interactions $|\mathbf{I}|$ is of the order $\mathcal{O}(|\mathbf{P}|^2) = \mathcal{O}(|\mathbf{\Lambda}|^4)$, since only interactions between pairs of particles are taken into account in the particle models. Not every arbitrary pair of particles can interact with each other, however. For example, when the particular domains between which two different particles form boundaries are different, these two particles could never collide

because of these domain conflicts. Also, when two different particles have the same velocities, they can never collide. On the other hand, some particle interactions will have multiple interaction results, which means that this particular interaction has to be stored multiple times, once for each possible outcome. Thus, in general, the number of particle interactions in the particle catalog is $\mathcal{O}(|\mathbf{\Lambda}|^4)$, each one requiring a constant amount of memory.

Finally, the current particle configuration needs to be stored while running a CA's particle model. Assume that the average number of particles occurring at t_c is n. Then $\mathcal{O}(n)$ memory is needed to store each subsequent particle configurations, since the number of particles after the condensation time can never increase while running the particle model (recall that the number of particles resulting from an interaction is limited to two).

Summing up all the memory requirements results in a space complexity of $\mathcal{O}(k^r + N + |\mathbf{\Lambda}|^4 + n)$. So, as with CAs, the particle models are in the space complexity class ExpSpace, or in PSpace when r is considered fixed, or when a more concise approximation (requiring a polynomial amount of space) of the PPD at t_c can be found.

Time complexity

The time complexity of a particle model depends partly on the kinds of particle interactions that can occur. It makes a difference, for example, whether all particle interactions create two new particles, or whether all particle interactions are annihilations. In trying to cover the whole range of possible time complexities for the particle models, both of these extremes are considered in the following complexity analysis. If the particle logic of a particular instance of the particle models is somewhere in between these two extremes (e.g., some particle interactions create two new particles, some only one new particle, and some are annihilations), then the time complexity of this model instance will also be somewhere in between those of the two extremes.

First assume a serial model of computation and the one extreme where every particle interaction creates two new particles. In step 1 in the algorithm in the particle models, an initial particle configuration at t_c is constructed. The time it takes to do this depends, of course, on the particular approximation of the PPD at t_c that is used. In the standard method of running the CA up to t_c and then copying the particle configuration, this will take $t_c N$ time steps, where N is again the lattice size. Assuming that the condensation time is bounded by some constant (a proof of this, in a simple setting, is given in chapter 6), this becomes a time complexity of $\mathcal{O}(N)$. Using more concise approximations of the PPD at t_c, as the ones presented in chapter 6, the time needed to construct an initial particle configuration is on the order of the number n of particles at t_c. In that case, the time complexity of this first step is $\mathcal{O}(n)$. Of course, in general $n \ll N$ (this will be addressed later on).

Step 2 of the algorithm, a check on the number of particles, only takes $\mathcal{O}(1)$ time. In step 3 the next interaction time t_i is calculated. Since this involves n simple algebraic calculations (as discussed in the previous section), this takes time $\mathcal{O}(n)$. In step 4 the two interacting particles are replaced with their interaction result (two new particles). This involves looking up the interaction result, which can be done in constant time, and then replacing the two interacting particles with the two resulting particles, which also takes constant time. Updating the position of the other particles in the lattice, the final part of step 4, takes $\mathcal{O}(n)$ time. Step 5 completes the main loop of the algorithm, obviously taking only time $\mathcal{O}(1)$. In the final step the positions of the particles are updated one more time. This again takes $\mathcal{O}(n)$ time, since the number of particles has neither increased nor decreased by assumption.

It appears that each step in the main loop of the algorithm (steps 2 to 5) takes at most $\mathcal{O}(n)$ time. The number of times this main loop is executed is equal to the number of particle interactions i that happen between the time steps t_c and M. In principle, this number can vary from $i = 0$ (no interactions at all) to $i = M$ (one interaction at each time step). To cover this wide range, a worst case, average case,

and best case analysis is done.

In the worst case, there is one interaction in each of the $M = 2N$ time steps. In that case, the main loop is executed $i = 2N = \mathcal{O}(N)$ times, taking time $\mathcal{O}(nN)$. In the best case, there are no interactions at all, for example because all particles travel with the same velocity. In that case, step 3 of the main loop is executed once, taking time $\mathcal{O}(n)$. In the average case, the n particles are $\mathcal{O}(N/n)$ sites apart, resulting in an interaction every $\mathcal{O}(N/n)$ time steps on average. This gives $i = \mathcal{O}(\frac{M}{N/n}) = \mathcal{O}(\frac{N}{N/n}) = \mathcal{O}(n)$ interactions on average. So, the main loop is executed $i = 2n = \mathcal{O}(n)$ times, taking time $\mathcal{O}(n^2)$.

Concluding, the time complexity of the particle models, assuming the one extreme where all particle interactions create two new particles, is anywhere between $\mathcal{O}(n)$ for the best case and $\mathcal{O}(nN)$ for the worst case, also depending on the approximation of the PPD at t_c that is used.

In the other extreme case, where all particle interactions are annihilations, the analysis is much easier. The construction of the initial particle configuration obviously still takes $\mathcal{O}(N)$ or $\mathcal{O}(n)$ time, depending on the method used. After that, all that needs to be done are the n algebraic calculations of the interaction times, taking $\mathcal{O}(n)$ time. Then the $n/2$ annihilations can be executed independently of each other, also taking $\mathcal{O}(n)$ time. So, for this extreme, the time complexity ranges from $\mathcal{O}(n)$ for the best case to $\mathcal{O}(N)$ for the worst case, depending on the approximation of the PPD at t_c that is used.

Summarizing, the serial-time complexity of a CA's particle model is anywhere between $\mathcal{O}(n)$ and $\mathcal{O}(nN)$, and thus it falls within the time complexity class P.

Moving on to a parallel model of computation using at most $\mathcal{O}(N)$ (or $\mathcal{O}(n)$) processors, the first step of creating a particle configuration at t_c can clearly be done in constant time. Furthermore, every step in the main loop of the algorithm, and also the final step, can be done in constant time. For example, calculating the interaction times for all neighboring pairs of particles can be done in parallel. Also, updating the

positions of all particles in the lattice can be done in parallel.

In the extreme case where all particle interactions create two new particles, the parallel time complexity is thus determined by the number i of times that the main loop of the algorithm needs to be executed. As was already calculated before, these numbers are $i = 0$, $i = 2N = \mathcal{O}(N)$, and $i = 2n = \mathcal{O}(n)$, for the best, worst, and average case, respectively. Consequently, the parallel time complexities in these cases are $\mathcal{O}(1)$, $\mathcal{O}(N)$, and $\mathcal{O}(n)$, respectively. For the other extreme of all annihilating interactions, the main loop only needs to be executed once, since all $n/2$ annihilations can be done independently of each other. Consequently, the parallel time complexity for this extreme case is simply $\mathcal{O}(1)$.

As with CAs, it appears that the particle models remain in the class P when moving from a serial to a parallel model of computation, at least for the worst and average cases of the one extreme where all particle interactions create two new particles. Again, this is not surprising given the fact that some one-dimensional CAs have been proved to be computationally universal by making explicit use of particles and their interactions. For example, in [LN90] several one-dimensional CAs are constructed that use particle-like structures (and their interactions) to simulate a Turing machine. Furthermore, it has recently been shown that the particles in ECA 110 can be used to simulate a certain set of production rules that is computationally universal [Coo00].

The fact that there exist particle logics that are computationally universal, implies directly that actually running such a particle logic is P-complete. For example, it becomes undecidable whether two arbitrary particles occurring in the IC will have interacted by time t. However, not all particle logics are computationally universal, and some are even efficiently parallelizable. For example, the parallel-time complexity of the one extreme of all annihilating particle interactions becomes $\mathcal{O}(1)$.

Summarizing, a CA's particle model is in the time complexity class P, as is the CA itself. However, the time complexity of a CA is $\mathcal{O}(N^2)$, whereas the time

complexity of a particle model is $\mathcal{O}(nN)$ or even $\mathcal{O}(n)$ in the best case. In chapter 6 it is shown that for the evolved CAs n is about two orders of magnitude smaller than N. So, in practice, a CA's particle model is run much faster than the CA itself, even though they are both in the complexity class P.

So, the time complexity of a particle model is lower than that of a CA. Furthermore, for some instances, running a particle model is efficiently parallelizable whereas running a CA is not. The space complexities of CAs and their particle models are roughly equal. In case of a concise approximation of the PPD at t_c, however, the memory requirements of a particle model might become slightly lower than that of a CA. Table 4.1 summarizes the results of the computational complexity analysis for CAs and their particle models.

Concluding, a CA's particle model is indeed a more concise description of the dynamics of a CA compared to actually running the CA itself. A particle model needs about the same amount of memory, or perhaps slightly less, than a CA. However, it takes significantly less time to run a particle model than to run a CA. In some cases, it is even possible to efficiently parallelize the particle model algorithm, which is not possible for running a CA.

		CA	Model $p_1 + p_2 \rightarrow p_3 + p_4$	Model $p_1 + p_2 \rightarrow \emptyset$
Space		$\mathcal{O}(k^r + N)$	$\mathcal{O}(k^r + N + \lvert\Lambda\rvert^4 + n)$	$\mathcal{O}(k^r + N + \lvert\Lambda\rvert^4 + n)$
Serial time	worst	$\mathcal{O}(N^2)$	$\mathcal{O}(nN)$	$\mathcal{O}(N)$
	average		$\mathcal{O}(n^2)$	
	best		$\mathcal{O}(n)$	$\mathcal{O}(n)$
Parallel time	worst	$\mathcal{O}(N)$	$\mathcal{O}(N)$	$\mathcal{O}(1)$
	average		$\mathcal{O}(n)$	
	best		$\mathcal{O}(1)$	

Table 4.1: The computational complexities of a CA and two extreme versions of a particle model. The number n of particles at t_c is about two orders of magnitude smaller than the lattice size N.

Chapter 5

Predicting the Computational Performance of Evolved Cellular Automata

In this chapter, the class of particle models, as introduced in the previous chapter, is used to analyze the relation between dynamics and emergent computation in evolved cellular automata. First, the particle models are used to predict quantitatively the computational performances of evolved CAs, based on their particle logics (i.e., emergent dynamics). Next, an error analysis is done, where it is shown how the simplifying assumptions in the class of particle models sometimes give rise to errors in simulating an evolved CA's computational strategy. The errors caused by the simplifying assumptions explain the (slight) discrepancies between the actual and predicted CA performances.

In addition to predicting a CA's performance, the particle models are also used to predict the time it takes a CA, on average, to settle down to an answer state. These time-to-answer predictions are then compared to the values as measured for the CA. For some CAs, there appear to be significant but consistent discrepancies between these predictions and the measured values. As with the performance predictions, it is shown how the simplifying assumptions in the model class lead to these discrepancies.

Finally, the class of particle models is used to determine quantitatively to what extent differences in the computational strategies of evolutionarily related CAs contribute to differences in their performances. In addition, the models are used to predict the performance of a particular evolved CA as a function of the relative frequencies of occurrences at t_c of the CA's particles. These predictions are then compared to analytically calculated values.

5.1 Performance predictions

In this section, the particle models are used to predict the performances of the evolved CAs that were shown in chapter 3. The performance measurements reported here are averages over 10 sets of 10,000 random ICs each. These ICs are chosen with a binomial distribution over ρ_0. A lattice size of $N = 149$ is used for the $\mathbf{T}_{dens}$ and $\mathbf{T}_{sync1}$ tasks

(as in the original evolving cellular automata experiments), and $N = 150$ for the $\mathbf{T}_{sync2}$ and $\mathbf{T}_{sync3}$ tasks (since these tasks need an even lattice size).

The particle catalogs of the evolved CAs analyzed here are presented in appendix A. The approximation of the PPD at t_c is as explained in the previous chapter: the CA being modeled is run up to t_c (starting with a random IC), and then the CA's particle configuration at t_c is copied to the lattice in the CA's model. The same 10 sets of 10,000 random ICs that are used for the calculation of the CA performances are used to generate initial particle configurations for a CA's particle model, and the predicted performances are averaged over these same 10 sets. Furthermore, for each IC the value of t_c (which depends on the IC) is used in the model together with the particle configuration at this t_c.

Figure 5.1 shows the performance measurements for the five CAs from the GA run on the density classification task. Space-time diagrams of these CAs (ϕ_{dens1} to ϕ_{dens5}) were shown in figure 3.3. In figure 5.1, the white bars represent the CA performances and the black bars represent the predicted performances.

Note that for the density classification task the correct answer to which the CA must settle down (the all-0s or the all-1s domain) depends on the density ρ_0 of the IC. Since the particle model of a CA is started with an initial particle configuration at t_c, and not with a random IC at $t = 0$, the information about the correct answer is not available. However, the initial particle configuration at t_c is copied from the CA after running it up to t_c starting from some random IC. So, when copying the particle configuration at t_c from the CA to the model, the information about the density ρ_0 of the IC can still be retrieved. This way, the correct answer (white domain or black domain) is known in the model, and the performance predictions can be calculated.

As figure 5.1 shows, the CAs' performances are lower than their corresponding fitness values as measured during the GA run (see figure 3.3). Since in a CA's performance measurement the cells in the lattice are assigned a 0 or a 1 with equal probability to create an IC, as explained earlier, the densities ρ_0 of the ICs are clus-

tered around 0.5. So, for the density classification task, this presents a more difficult set of ICs to classify than with the fitness measurement (where the densities ρ_0 are uniformly distributed between 0.0 and 1.0). Therefore, the performance of a density classification CA is generally lower than its fitness value.

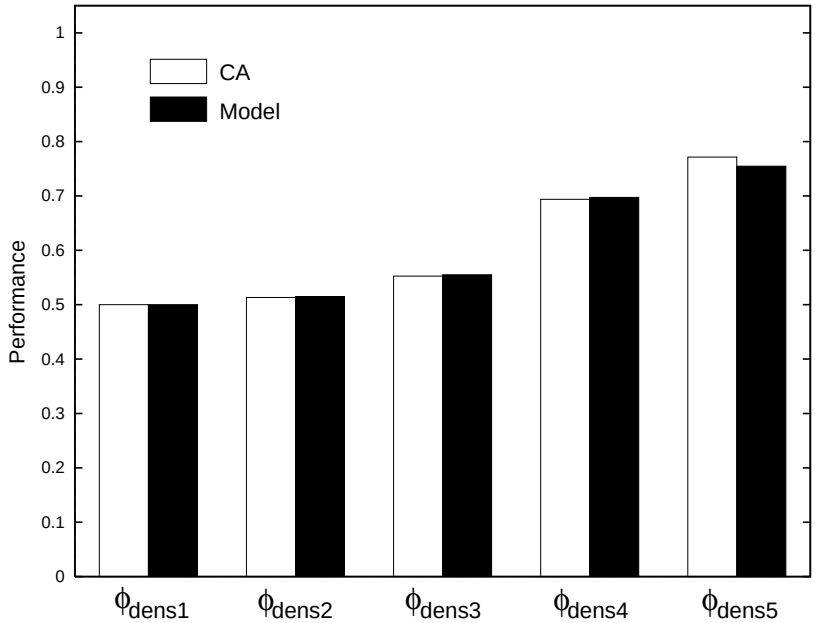

Figure 5.1: CA and particle model performances for the five CAs from the GA run on the density classification task.

As figure 5.1 shows, the agreements between the CA performances and those predicted by their particle models are excellent. Table 5.1 presents the numerical values, including standard deviations and the percentages of difference between CA and model performances. As the table shows, most differences are within 0.5%, except for ϕ_{dens5}, for which the difference is 1.6%. In section 5.2 below, some examples are given of how these (small) discrepancies are caused by the simplifying assumptions in the model class.

This first set of results shows that the class of particle models is indeed capable of accurately predicting the performance of evolved CAs. This gives support to the claim that the particle-level description of a CA's dynamics captures the main

	CA	Model	% error
ϕ_{dens1}	0.5000 (0.0000)	0.5000 (0.0000)	0.0
ϕ_{dens2}	0.5137 (0.0023)	0.5138 (0.0022)	0.0
ϕ_{dens3}	0.5485 (0.0029)	0.5507 (0.0027)	0.4
ϕ_{dens4}	0.6923 (0.0042)	0.6954 (0.0043)	0.4
ϕ_{dens5}	0.7724 (0.0047)	0.7600 (0.0047)	1.6

Table 5.1: Performance measurements for the five CAs from the density classification run. The first column gives the CA performance, the second column gives the predicted performance, and the last column shows the percentage difference between the CA and predicted performances. Standard deviations for the performance measurements are given in parentheses.

mechanisms of the CA's emergent computation necessary to perform a given task. Next, performance prediction results for the CAs evolved on the other computational tasks are presented.

Figure 5.2 shows performance prediction results for the five CAs from the GA run on the global synchronization–1 task. Space-time diagrams of these CAs (ϕ_{sync1} to ϕ_{sync5}) were shown in figure 3.4. In figure 5.2, again, the white bars represent the CA performances and the black bars represent the model performances. The performance of ϕ_{sync1} is actually 0.0, so the bars don't show up in this plot. ϕ_{sync1}'s particle model correctly predicts a performance of 0.0. Note that the performances of these evolved CAs are also lower than their corresponding fitness values as measured during the GA run (see figure 3.4). This is for the same reason as for the density classification CAs, i.e., performance measurements present a more difficult set of ICs than fitness measurements.

For the global synchronization tasks (all three versions), the correct answer does not depend on some property of the IC. There is only one correct answer state

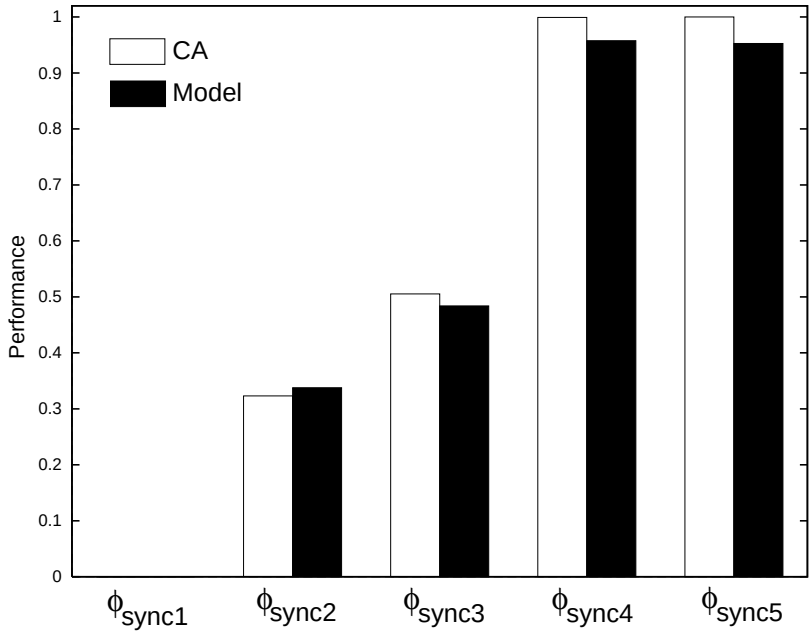

Figure 5.2: CA and particle model performances for the five CAs from the GA run on the global synchronization–1 task.

(the globally synchronized one), regardless of what the IC is. So, for these tasks, no additional information is necessary in a CA's particle model when starting with an initial particle configuration at t_c (generated by the approximation of the PPD at t_c).

As figure 5.2 shows, the agreements between the CA performances and those predicted by their particle models are still good, although the discrepancies are slightly larger than for the density classification CAs. Table 5.2 presents the numerical values, again including standard deviations and the percentages difference between CA and model performances. In this case, the differences are all within a 5% error margin.

The somewhat larger discrepancies for these five synchronization CAs are partly due to the fact that some of these CAs (in particular ϕ_{sync4} and ϕ_{sync5}) have particle interactions that can have multiple outcomes depending on the relative phases that the interacting particles are in at the time of collision. Since in the particle models this is approximated by a stochastic choice of interaction result, this

can give rise to certain particle configurations that can not occur in the CA itself. Examples of this are given in section 5.2 below.

	CA	Model	%
ϕ_{sync1}	0.0000 (0.0000)	0.0000 (0.0000)	0.0
ϕ_{sync2}	0.3230 (0.0048)	0.3377 (0.0036)	4.6
ϕ_{sync3}	0.5052 (0.0026)	0.4839 (0.0025)	4.2
ϕ_{sync4}	0.9992 (0.0004)	0.9577 (0.0014)	4.2
ϕ_{sync5}	1.0000 (0.0000)	0.9527 (0.0016)	4.7

Table 5.2: Performance measurements for the five CAs from the synchronization–1 run. The columns are the same as in table 5.1.

Finally, figure 5.3 shows performance prediction results for the four CAs that were evolved on the two new tasks, the global synchronization–2 and the global synchronization–3 tasks, respectively. Space-time diagrams of these CAs ($\phi_{\text{sync-2a}}$, $\phi_{\text{sync-2b}}$, $\phi_{\text{sync-3a}}$, and $\phi_{\text{sync-3b}}$) were shown in figure 3.9. In figure 5.3, again, the white bars represent the CA performances and the black bars represent the model performances.

On these new tasks too, the agreements between the CA performances and those predicted by their particle models are excellent. Table 5.3 presents the numerical values, again including standard deviations and the percentages difference between CA and model performances. As with the density classification CAs, most differences are within 0.5%, except for $\phi_{\text{sync-2b}}$, for which the difference is 3.1%. Again, for these four CAs, the performance is lower than the respective fitness values (see section 3.6).

In summary, generally there is a very good agreement between an evolved CA's performance and that predicted by its particle model: differences are often within 0.5%. In some cases, particularly for the CAs that were evolved for the global

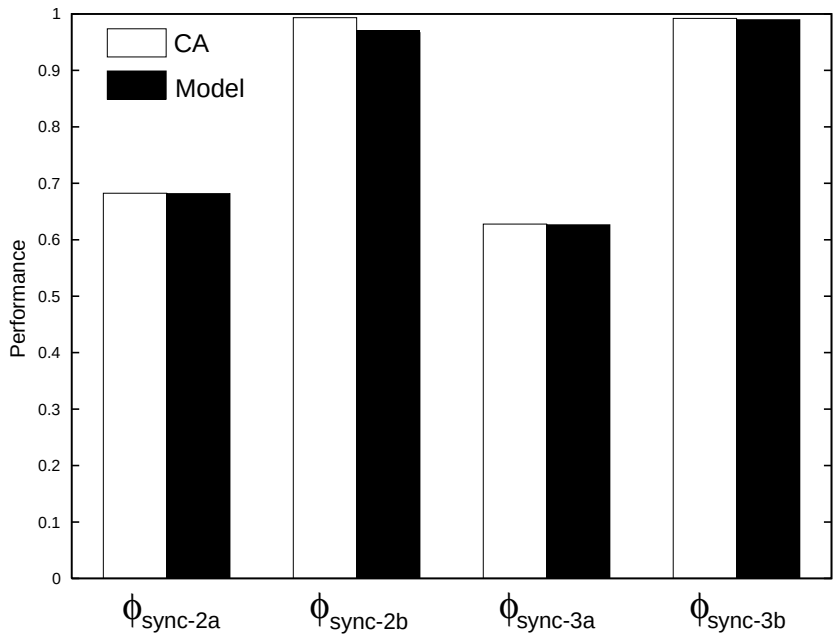

Figure 5.3: CA and particle model performances for the two CAs from the GA run on the global synchronization–2 task, and the two CAs from the GA run on the global synchronization–3 task.

synchronization–1 task, the discrepancies are (slightly) larger, but still within 5%. As discussed below, these discrepancies are due to the simplifying assumptions incorporated in the class of particle models. Section 5.2 gives some examples of how these assumptions can cause a difference between a CA's dynamics and that of its particle model.

5.2 Error analysis

As the results in the previous section show, an evolved CA's particle model accurately predicts the CA's performance, based on a description of the CA's dynamics at the particle-logic level. In other words, a CA's particle model is capable of accurately simulating that aspect of a CA's dynamics that gives rise to its computational performance on a given task. Figure 5.4 shows an example for $\phi_{\text{sync-3b}}$ of how the particle model accurately simulates the CA's dynamics at the level of the particles

	CA	Model	%
$\phi_{\text{sync-2a}}$	0.6825 (0.0049)	0.6818 (0.0042)	0.1
$\phi_{\text{sync-2b}}$	0.9935 (0.0008)	0.9632 (0.0012)	3.1
$\phi_{\text{sync-3a}}$	0.6277 (0.0049)	0.6264 (0.0050)	0.2
$\phi_{\text{sync-3b}}$	0.9922 (0.0008)	0.9897 (0.0007)	0.3

Table 5.3: Performance measurements for the four CAs for the synchronization–2 and synchronization–3 tasks. The columns are the same as in table 5.1.

and their interactions. The space-time diagram on the left shows the CA behavior starting from a random IC. The space-time diagram on the right shows the behavior as simulated by $\phi_{\text{sync-3b}}$'s particle model starting with a copy of the CA's particle configuration at t_c (which occurs at time step 38 in this particular example).

In some cases, however, the simplifying assumptions incorporated in the class of particle models cause a difference between the simulated behavior and the CA behavior. This difference can lead to a different answer state in a CA's model, compared to the answer state that the CA settles down to. This, in turn, leads to the observed discrepancies between a CA's actual and predicted performances.

For example, the performance of ϕ_{dens2} on one particular set of 10,000 random ICs is 0.5134. ϕ_{dens2}'s particle model gives a predicted performance of 0.5147, i.e., a difference of 0.0013. It turns out that there are 13 out of 10,000 ICs on which ϕ_{dens2} settles down to an incorrect answer state, but on which its particle model predicts that it will settle down to the correct answer state. This gives rise to a predicted performance that is slightly higher than the CA performance.

Figure 5.5 shows an example of one of those 13 cases for which the model diverges from the CA behavior. The space-time diagram on the left shows the CA, while the diagram on the right shows the result generated by the model, starting

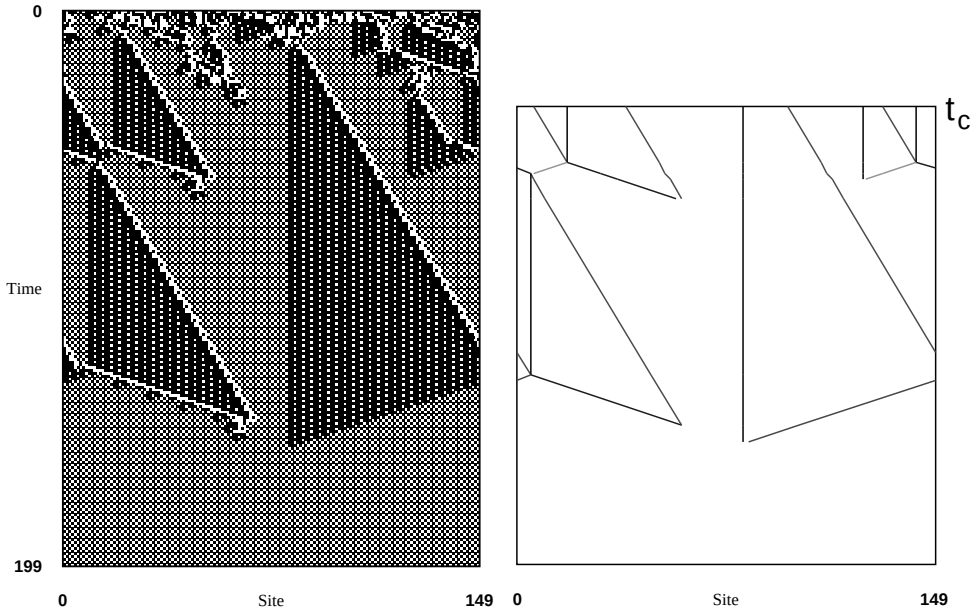

Figure 5.4: Comparison of a CA space-time diagram (left) with that generated by the particle model (right) for $\phi_{\text{sync-3b}}$. The model accurately simulates the CA's particle dynamics.

with a copy of the particle configuration at t_c. In the area marked by the circle, three particles, α, β, and γ, come close together (see ϕ_{dens2}'s particle catalog in appendix A for the different particle types). In the CA, particles α and β collide first and quickly annihilate each other. Therefore, the γ particle simply continues on its way and interacts with another β particle later on. In the model, however, the particles are assumed to have zero width. Because of this assumption, and given the relative positions of the three particles, the two particles β and γ collide first. The resulting interaction gives rise to another particle that travels faster than the α particle and, therefore, this α particle continues on its way. This causes the particle model to eventually settle down to a different answer state than the CA. The other 12 cases are similar.

Calculating the performance of ϕ_{dens3} on the same set of 10,000 ICs gives 0.5526, while its predicted performance on this set of ICs is 0.5542; a difference of 0.0016. In this case, there are 6 ICs on which the CA gives the correct answer, while

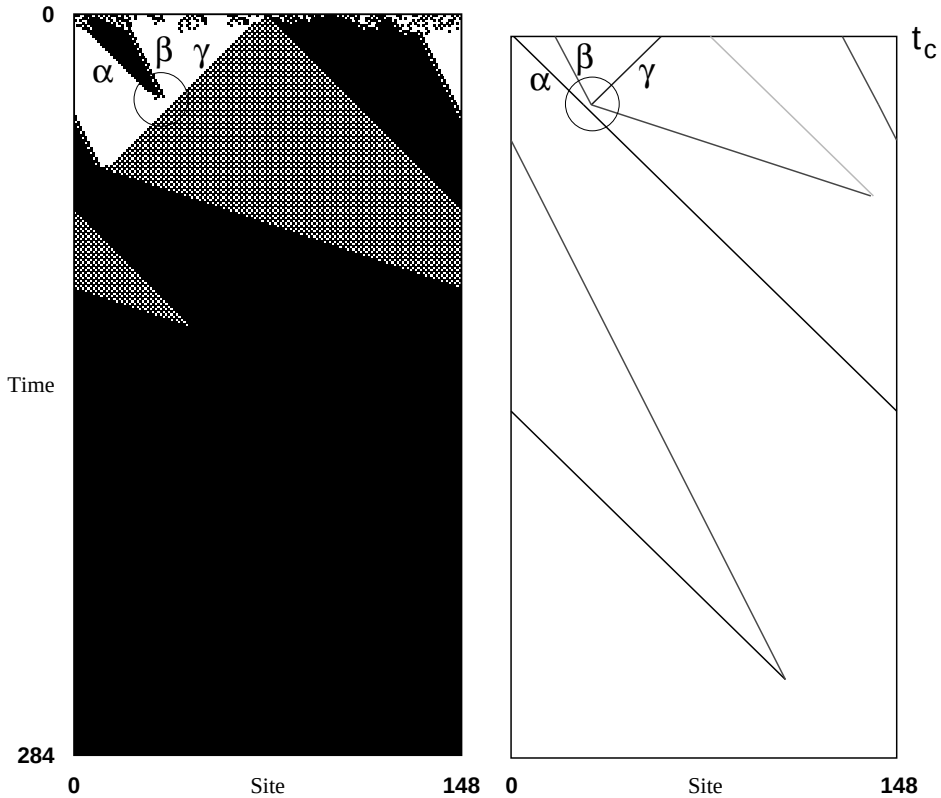

Figure 5.5: Comparison of a CA space-time diagram (left) with that generated by the particle model (right) for ϕ_{dens2}. The circle indicates the area where there is a difference between the CA and its model, which eventually leads to a different overall outcome.

the model predicts it will give the wrong answer. Similarly, there are 22 ICs on which the CA gives the wrong answer, while the model predicts it will give the correct answer. Again, these differences are caused by the simplifying assumptions, and give rise to the difference of $\frac{22-6}{10{,}000} = 0.0016$ between the CA and model performances.

An example of how the zero-width-particles assumption can cause some of these differences was shown in the space-time diagram of ϕ_{dens2} in figure 5.5. Figure 5.6 shows an example for ϕ_{dens5} of how the assumption of only pairwise interactions can cause similar differences. In the space-time diagram on the left (the CA), there is a three-particle collision in the circled area. This leads to a result of four new particles. In the space-time diagram on the right (the model), only two of these particles actually collide, and the third particle continues on its way without interacting with the other

two particles or with their interaction result. This eventually causes a different overall answer state.

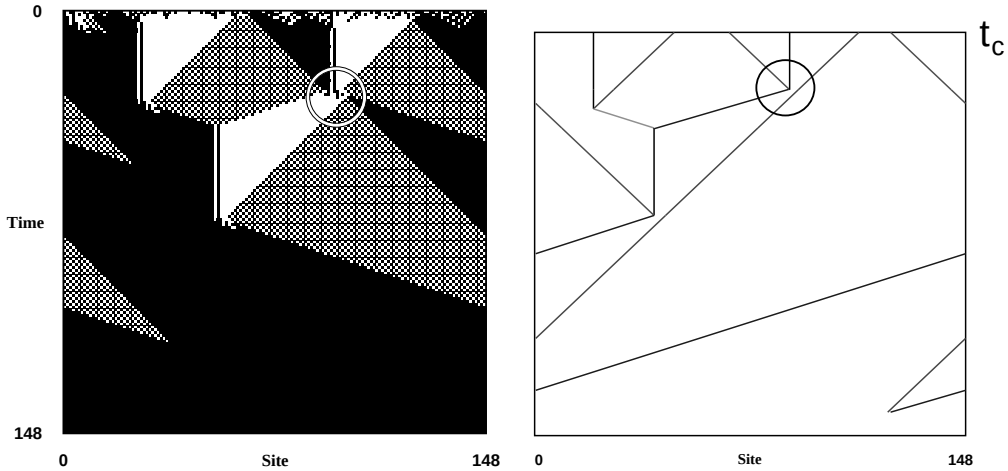

Figure 5.6: Comparison of a CA space-time diagram (left) with that generated by the particle model (right) for ϕ_{dens5}. The circle indicates the area where there is a difference between the CA and its model, which eventually leads to a different overall outcome.

Figure 5.7 shows an example, again for ϕ_{dens5}, of how the instantaneous interactions assumption causes differences. In the space-time diagram on the left (the CA), the interaction indicated by the circle results in one new particle of type ε. However, this particle appears a few time steps after the interacting particles collided, and slightly to the right of the location of this initial collision. This ε particle then interacts with a δ particle, which results in a β particle with zero velocity. There are two other particles, η and γ, moving towards this zero-velocity β particle with equal but opposite velocities. In this case, the γ particle approaching from the right reaches the β particle first, and consequently interacts with it. The η particle approaching from the left simply continues on its way.

In the space-time diagram on the right (the model), however, the ε particle that results from the interaction indicated by the circle is placed at the exact site where the two interacting particles collide, at the time of collision. In other words, it is slightly to the left of where the corresponding ε particle in the CA space-time diagram is.

As a consequence, the zero-velocity β particle resulting from the interaction between this ε particle and the δ particle is also slightly to the left of the corresponding β particle in the CA space-time diagram. This causes the β particle to interact with the η particle approaching from the left, instead of the γ particle approaching from the right, which now continues on its way. This, again, leads to a different overall outcome.

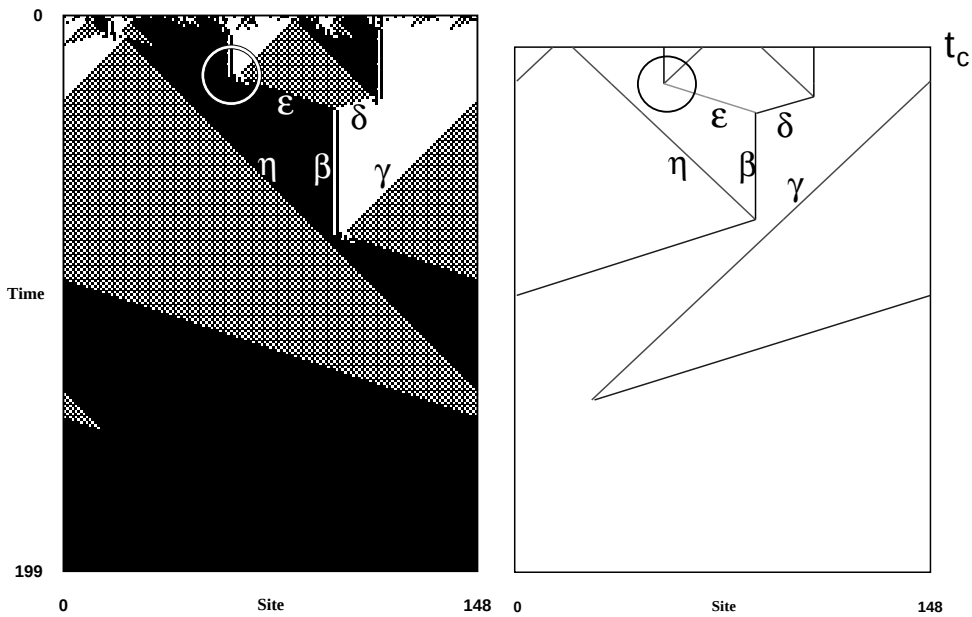

Figure 5.7: Comparison of a CA space-time diagram (left) with that generated by the particle model (right) for ϕ_{dens5}. The labels indicate the particles that cause a difference in the overall outcome.

The performance of ϕ_{dens5} on one set of 10,000 ICs is 0.7714, while its model performance on this set of ICs is 0.7606. This difference of 0.0108 is caused by 211 ICs on which the CA correctly classifies the initial density, while the model predicts otherwise, and 103 ICs on which the CA gives the wrong answer, while the model predicts a correct answer.

How the final simplifying assumption, a stochastic choice of interaction result, causes differences is illustrated with ϕ_{sync5} in figure 5.8. The space-time diagram on the left shows the CA, while the space-time diagram on the right shows the result

generated by ϕ_{sync5}'s particle model. In the area indicated by the circle, a β and a γ particle collide. This particular interaction can have two different outcomes (see ϕ_{sync5}'s particle catalog in table 3.2): $\delta + \mu$ with probability 0.84, and ν with probability 0.16. In the CA, the interaction result is determined by the relative phases that the β and γ particles are in at the time of collision. In ϕ_{sync5}'s particle model, however, the result is chosen at random according to the indicated probabilities. In fact, running the model several times starting with the same particle configuration at t_c can give rise to a different particle behavior each time, exactly because of this stochastic choice.

In the space-time diagram on the right in figure 5.8, the interaction result ν was chosen for the interaction indicated by the circle. Since at the time of this interaction there was one other particle present in the lattice, also a ν particle, there are now two ν particles left in the lattice. Since these particles are of the same type, they have the same velocities, and thus never interact with each other. In other words, these two particles will remain in the lattice forever, with zig-zag domains Λ^z in between them. Thus, the globally synchronized state will never be reached.

Since the CA is iterated on a lattice of size $N = 149$, a configuration consisting of two ν particles cannot fit on the lattice due to the spatial periodicity $p^s_{\Lambda^z} = 4$ of the zig-zag domain and the periodic boundary conditions. However, in the CA's particle model this spatial periodicity is not taken into account, and thus this constraint does not exist. As it turns out, on roughly 5% of the ICs, ϕ_{sync5}'s particle model does generate such a configuration because of the stochastic choice of interaction results. The performance of ϕ_{sync5} on the set of 10,000 ICs is 1.0000, while its predicted performance is only 0.9522. Indeed, there are 478 ICs (out of the 10,000) on which the model creates a "forbidden" configuration of two ν particles.

So far, examples have been given only of how the simplifying assumptions in the class of particle models can cause differences that lead to different final outcomes between a CA and its model. However, it is also possible that differences caused by

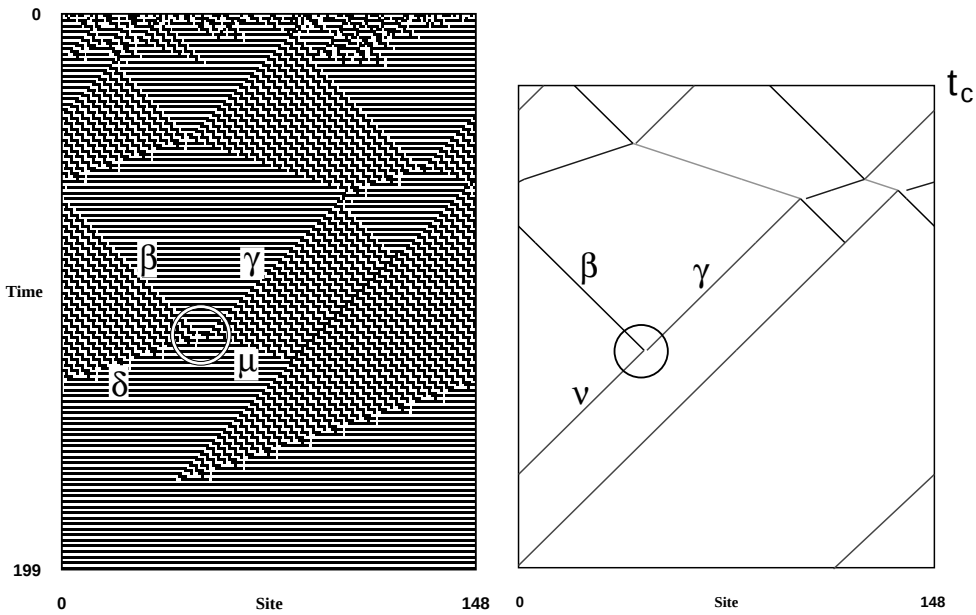

Figure 5.8: Comparison of a CA space-time diagram (left) with that generated by the particle model (right) for ϕ_{sync5}. The circle indicates the area where there is a difference.

the simplifying assumptions do not cause any difference in the final outcome. Or, said another way, two differences in the model caused by the simplifying assumptions can cancel each other out and still lead to the same final outcome as in the CA.

Figure 5.9 shows an example of this, again for ϕ_{sync5}. The space-time diagram on the left shows the CA, where two particle interactions are highlighted by circles. The corresponding space-time diagram generated by ϕ_{sync5}'s model is shown on the right, with the same two interactions highlighted. As the figure shows, for both these interactions, the model chose a different interaction outcome compared to what happened in the CA. However, this still leads the model to settle down to the globally synchronized state, although in a slightly different manner than in the CA.

In fact, when ϕ_{sync5}'s model is run again with the exact same particle configuration at t_c, it can generate a different space-time diagram, since it can choose different results for the same interactions. However, when predicting a CA's performance with a particle model, these "statistical" differences are averaged out. In other words, the CA's particle model is still capable of accurately predicting its performance.

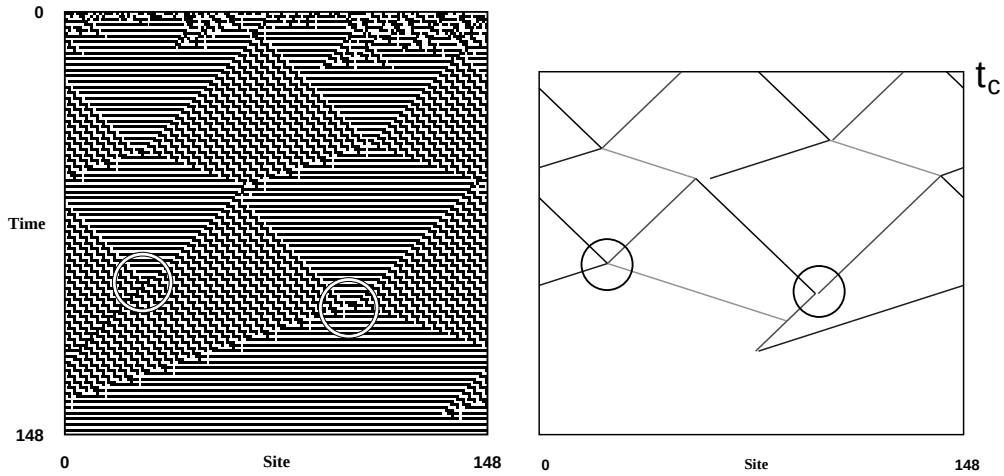

Figure 5.9: Comparison of a CA space-time diagram (left) with that generated by the particle model (right) for ϕ_{sync5}. The circles indicate the areas where there is a difference.

5.3 Time-to-answer predictions

The performance of a CA is determined by the fraction of ICs on which it settles down to the correct final configuration. However, not only must the CA settle down to the correct answer state, it also must do this within the maximum number M of allowed time steps. So, the *time-to-answer* t_a of a CA is an important statistic that partly determines its performance.

The class of particle models can be used to make predictions about t_a for the evolved CAs. Given a CA and a computational task, an average time-to-answer $\overline{t_a}$ is calculated as the average, over a random sample of ICs, of the number of time steps it takes the CA to settle down to an answer state, regardless of whether this answer state is the correct one. In this average only ICs are included on which the CA indeed settles down to an answer state within M time steps. A CA's particle model is then used to predict this $\overline{t_a}$ by calculating the same average time to settle down to an answer state in the CA's model, calculated over a random sample of particle configurations at t_c (generated by the actual CA).

Table 5.4 presents the CA and predicted $\overline{t_a}$ values for the five density classi-

fication CAs and the five global synchronization–1 CAs analyzed earlier. The measurements are averaged over the same 10 sets of 10,000 ICs that were used for the performance measurements and predictions in section 5.1. Recall that the performance of ϕ_{sync1} is 0.0, which means that it never settles down to the (one and only) answer state, and thus there is no $\overline{t_a}$ value for this CA.

	CA	Model			CA	Model
ϕ_{dens1}	54 (0.44)	71 (0.42)		ϕ_{sync1}	–	–
ϕ_{dens2}	152 (0.76)	158 (0.77)		ϕ_{sync2}	186 (1.32)	192 (1.23)
ϕ_{dens3}	71 (0.48)	73 (0.49)		ϕ_{sync3}	116 (1.00)	112 (1.01)
ϕ_{dens4}	87 (0.25)	89 (0.27)		ϕ_{sync4}	92 (0.48)	87 (0.54)
ϕ_{dens5}	84 (0.25)	85 (0.27)		ϕ_{sync5}	87 (0.46)	82 (0.47)

Table 5.4: CA and model-predicted $\overline{t_a}$ values, averaged over 10 sets of random ICs. Standard deviations are given in parentheses.

As the table shows, there are rather large discrepancies between the CA and predicted values for some CAs. Note that, especially for the density classification CAs, the predictions get better for the CAs that occurred later on in the evolution.

It turns out that the main reason for the discrepancies here is the zero-width particle assumption. Figure 5.10 shows an example of this for ϕ_{dens1}, for which the discrepancy is the largest. At time step $t = 0$, there are two particles present in the lattice: ε at site $i = 10$ and γ at site $i = 20$. The ε particle travels with a velocity of $v_\varepsilon = 4/3$ and the γ particle with a velocity of $v_\gamma = 1$ (see ϕ_{dens1}'s particle catalog in appendix A). The two particles start interacting as soon as they are within each other's local neighborhood (recall that $r = 3$ in this CA). By time step $t = 7$ they have annihilated each other, as indicated by the first dotted line.

In ϕ_{dens1}'s particle model, however, these particles have no width and travel as

indicated by the solid white lines in figure 5.10. Their interaction time t_i is calculated as follows:

$$\begin{aligned} 10 + 4/3t_i &= 20 + t_i \quad \Rightarrow \\ t_i &= 30 \end{aligned}$$

So, only at time step $t = 30$ will these two particles collide and subsequently annihilate each other (indicated by the second dotted line), as compared to time step $t = 7$ in the actual CA.

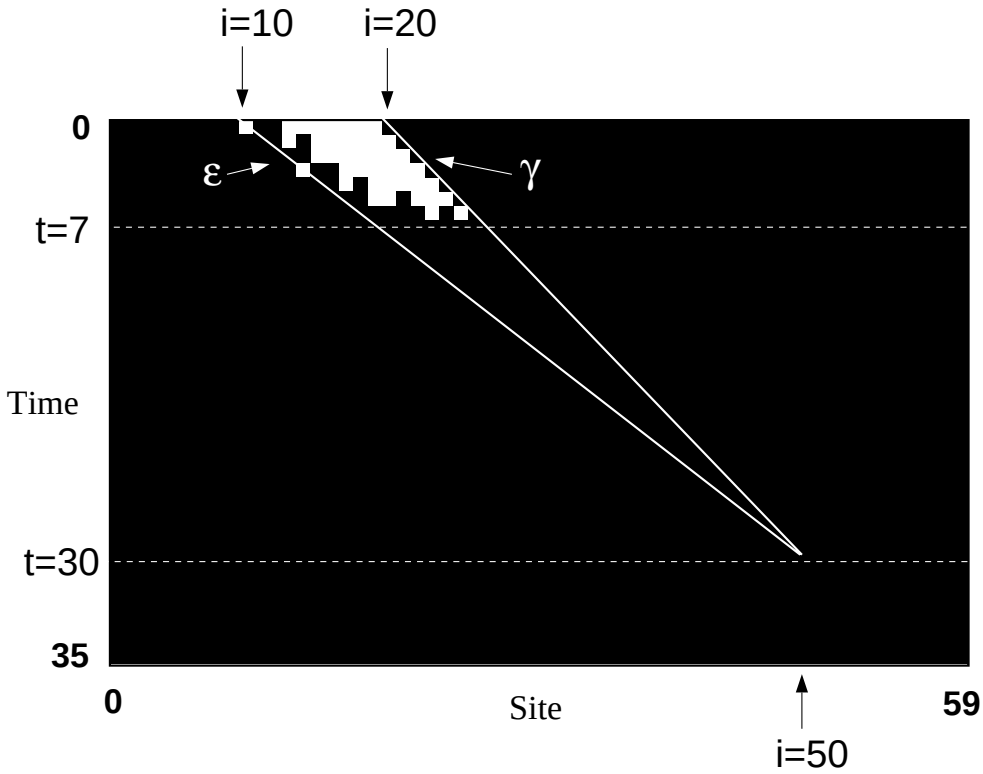

Figure 5.10: Example of the difference in interaction time t_i between the CA and its particle model.

Figure 5.10 clearly shows why ϕ_{dens1}'s particle model predicts a larger $\overline{t_a}$ than the CA value: it takes longer for the particles in the model to annihilate each other than in the CA. This effect is less extreme in the other density classification CAs, and their respective particle models indeed give a better prediction. For several

synchronization CAs, however, an opposite effect occurs. For these CAs, there are particle interactions that temporarily create some non-domain/particle configurations before they completely annihilate each other. This can be seen clearly, for example, in figure 4.2. When this occurs, the corresponding particles in the models annihilate slightly earlier, because of the instantaneous interaction assumption in the model class. This mechanism explains the lower $\overline{t_a}$ predictions as compared to the CA values for several of the synchronization CAs.

Summarizing, the class of particle models is capable, within some margin of error, to predict the time it takes on average for a CA to settle down to an answer state. The discrepancies between the predictions and the CA values are mainly due to two of the simplifying assumptions in the model class: zero-width particles and instantaneous interactions.

5.4 Comparative analysis

In addition to predicting a CA's performance and the time it takes on average to settle down to an answer state, the class of particle models can be used to determine, in a quantitative way, to what extent changes in a CA's particle logic contribute to changes in its performance. For example, previously it was claimed that the higher performance (or fitness) of ϕ_{dens4}, compared to that of ϕ_{dens3}, was due to the change in velocity of the β particle from 1/3 to 0 [DMC94].

Until the development of the class of particle models, this claim could not be verified. In other words, in a CA it is practically impossible to separate out the properties of a particle (e.g., its velocity or the way it interacts with other particles), and study these properties in isolation leaving everything else in the CA the same (see also the discussion in chapter 7). However, in the class of particle models this is possible. For example, the velocity of a certain particle can be changed, leaving everything else the same. It can then be investigated what effect this change in

particle velocity has on the CA's performance.

In this section, the class of particle models is used as a comparative analysis tool to investigate the above claim. Furthermore, other pairs of evolutionarily related CAs are analyzed to determine to what extent differences in their particle logics contribute to differences in their performances. In total, four cases are analyzed in this section, using the particle models.

5.4.1 Case 1: $\phi_{\mathbf{dens3}}$ and $\phi_{\mathbf{dens4}}$

The particle catalogs for ϕ_{dens3} and ϕ_{dens4} are given in appendix A. As these catalogs show, the particle logics of these two evolutionarily related CAs are exactly the same except for the velocity of the β particle. In ϕ_{dens3}, this velocity is $v_\beta = 1/3$, while in ϕ_{dens4} it is $v_\beta = 0$. However, there is a significant difference in their performances (about 0.14), as figure 5.1 and table 5.1 show. Is this difference due only to the difference in the velocity of the β particle, as previously claimed?

Using the particle models, and one particular (fixed) set of 10,000 random ICs on which the CAs are run up to t_c to get an initial particle configuration at t_c, the performance of ϕ_{dens3} is predicted to be 0.5510 and that of ϕ_{dens4} to be 0.6928, i.e., a difference of 0.1418. Changing the velocity of the β particle from 1/3 to 0 in ϕ_{dens3}'s particle catalog, the performance of ϕ_{dens3} is now predicted to be 0.5693, i.e., an increase of only 0.0183. In other words, the change in velocity of the β particle from 1/3 to 0 contributes only about 13% to the total difference in performance between ϕ_{dens3} and ϕ_{dens4}.

Therefore, there must be at least one additional factor that contributes to this difference in performance. In terms of the particle models, the only other difference between ϕ_{dens3} and ϕ_{dens4} are the particle configurations at t_c, i.e., the approximations of the PPD at t_c. Since ϕ_{dens3} and ϕ_{dens4} have the exact same set of particle types, these approximations of the PPD at t_c can be interchanged between the two CAs, and the approximation of one CA can be used in the particle model of the

other CA. This way, it can be investigated how the difference between these two approximations contributes to the differences in the performances of these CAs.

Using the particle catalog of ϕ_{dens3} (with $v_\beta = 1/3$), and the approximation of the PPD at t_c of ϕ_{dens4}, the performance prediction is 0.6558, i.e., an increase of 0.1048. Now changing, in addition, the velocity of the β particle from 1/3 to 0, yields a predicted performance of 0.6928, i.e., ϕ_{dens4}'s predicted performance. This is not surprising, since the model now actually uses both the approximation of the PPD at t_c and the particle catalog of ϕ_{dens4}, and thus effectively *is* ϕ_{dens4}'s particle model. Indeed, just changing the velocity of the β particle from 0 to 1/3 in ϕ_{dens4}'s particle catalog, and using its own approximation of the PPD at t_c, also gives a predicted performance of 0.6558.

In short, using the respective particle models of ϕ_{dens3} and ϕ_{dens4}, one can go back and forth between using the β velocity or the approximation of the PPD at t_c from one or the other CA. From these prediction results it can be determined to what extent these factors contribute to the difference in performances between these two CAs. Table 5.5 summarizes the results of this analysis.

	PPD at t_c	
	ϕ_{dens3}	ϕ_{dens4}
$v_\beta = 1/3$	0.5510	0.6558
$v_\beta = 0$	0.5693	0.6928

Table 5.5: Performance predictions for ϕ_{dens3} and ϕ_{dens4}, interchanging their respective approximations of the PPD at t_c and velocities of the β particle.

As this analysis shows, the change in the velocity of the β particle makes only a small contribution (about 13%) to the difference in performances, unlike what was claimed previously. In fact, the main contributing factor is the change in the PPD at t_c. Figure 5.11 illustrates this change. The graph in this figure shows the frequency distribution of the total number of particles occurring at t_c, measured over the set of 10,000 random ICs used for the performance predictions presented in this section.

The bars show the frequency distribution for ϕ_{dens3}, while the dots connected by solid lines show the distribution for ϕ_{dens4}. For example, for ϕ_{dens3} on about 30% of the ICs there are two particles at t_c. For ϕ_{dens4}, however, this occurs on only about 20% of the ICs. On average, there are 3.2 particles at t_c for ϕ_{dens3} and 2.9 for ϕ_{dens4}. An analysis of how changes in the PPD at t_c cause changes in performance is presented for ϕ_{sync2} in case 3 below.

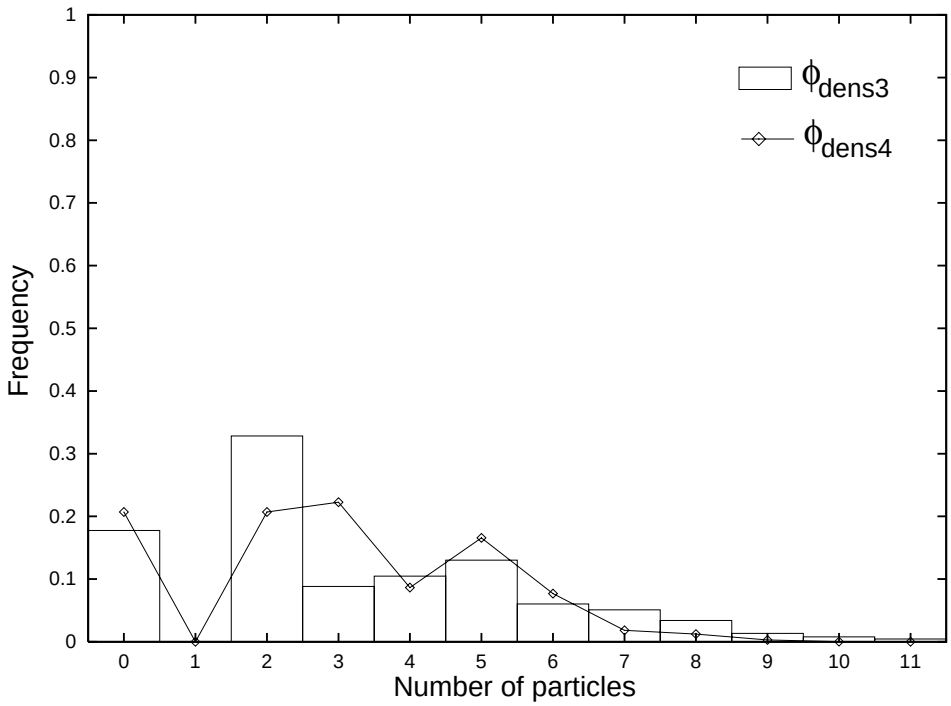

Figure 5.11: The frequency distributions of the total number of particles at t_c for ϕ_{dens3} and ϕ_{dens4}.

As a final remark, the results from this comparative analysis show that the two factors that contribute to the difference in performance between ϕ_{dens3} and ϕ_{dens4}, do so in a nonlinear way. On the one hand, using the PPD at t_c of ϕ_{dens3}, but changing the velocity of the β particle from 1/3 to 0, gives an increase in performance of 0.0183. On the other hand, using the PPD at t_c of ϕ_{dens4}, but leaving the velocity of the β particle at 1/3, gives an increase in performance of 0.1048. Adding these two numbers, assuming that these two effects work independently, gives an increase

in performance of only 0.1231. However, the actual increase in performance is 0.1418. So, the two changes do not work independently, but reinforce each other nonlinearly.

5.4.2 Case 2: ϕ_{dens4} and ϕ_{dens5}

Going from ϕ_{dens4} to ϕ_{dens5} there is again an increase in performance. The difference in the particle logics of these two CAs is that ϕ_{dens4}'s α particle (a boundary between a white domain and a black domain) does not exist in ϕ_{dens5}. Instead of an α particle, two particles (γ and ν) with a checkerboard domain in between them are created whenever a white domain borders a black domain in ϕ_{dens5}.

Using the particle models, and again one (fixed) set of 10,000 random ICs on which the CAs are run up to t_c to get a particle configuration at t_c, the performance of ϕ_{dens4} is predicted to be 0.6928 (as in case 1 above), and that of ϕ_{dens5} to be 0.7606, i.e., a difference of 0.0678. Now using the particle catalog of ϕ_{dens4}, but the approximation of the PPD at t_c of ϕ_{dens5} (which lacks α particles), the predicted performance is 0.7606, i.e., exactly ϕ_{dens5}'s predicted performance.

In conclusion, in this case the difference in performances between the two evolutionarily related CAs can be attributed completely to the differences in the PPD at t_c. In particular, eliminating the occurrences of α particles at t_c, and replacing them with a pair of γ and ν particles with a checkerboard domain in between, significantly improves the computational strategy of the CA, which is reflected in the corresponding increase in performance.

5.4.3 Case 3: ϕ_{sync2}

The class of particle models can be used to study in more detail how changes in the PPD at t_c cause a difference in a CA's performance. ϕ_{sync2} is used here to illustrate this. As ϕ_{sync2}'s particle catalog (presented in appendix A) shows, there are only two particle types, α and β, for this CA. Both these particles form a boundary between two synchronized domains Λ^s, but the domains on either side of a particle are out

of phase with each other. In other words, when at some time step t the domain on the left of a particle is in phase 0 (the all-0s configurations), then the domain on the right of the particle is in phase 1 (the all-1s configuration). When an α and a β particle collide, however, they annihilate each other, resolving these phase conflicts. Consequently, ϕ_{sync2} will only settle down to the globally synchronized state when the α and β particles occur in equal numbers at the condensation time, so they will all annihilate each other and resolve all existing domain phase conflicts.

In this case study it is investigated to what extent changes in the probability that a particle at t_c is of type α give rise to changes in ϕ_{sync2}'s performance. Given the probability $\Pr[n]$ of a total number n of particles at t_c, let p be the probability that a particle at t_c is of type α. In other words, $p = \Pr[\alpha|n]$ is a conditional probability, conditioned on n. The probability that the α and β particles occur in equal numbers at t_c, given n total particles, is the probability that $n/2$ of the n particles are of type α:

$$\Pr[n/2\ \alpha\text{'s}|n] = p^{n/2}(1-p)^{n/2}\binom{n}{n/2}$$

The CA's performance $\mathcal{P}$ is then the sum, over all even values of n, of the probability of n particles at t_c times the probability of $n/2$ α particles given n total particles:

$$\mathcal{P} = \sum_{n=0,2,\ldots} \Pr[n] \times p^{n/2}(1-p)^{n/2}\binom{n}{n/2}$$

For the probabilities $\Pr[n]$, the empirically observed frequencies of the total number n of particles at t_c for ϕ_{sync2} can be used. Taking again the fixed set of 10,000 random ICs, these frequencies are measured as shown in table 5.6. Using the above expression for the performance $\mathcal{P}$, together with these empirically measured probabilities, the performance of ϕ_{sync2} as a function of p can now be calculated. Table 5.7, second column, shows the result of this calculation for several values of p. Note that the performances are the same for p and $1-p$, since the expression is symmetric in p around 0.5.

n	$\Pr[n]$
0	0.0302
2	0.3621
4	0.4769
6	0.1235
8	0.0073
≥ 10	0.0000

Table 5.6: Empirically measured probabilities $\Pr[n]$ of the total number n of particles at t_c for ϕ_{dens2}.

To see how well these calculations compare with the real performances of ϕ_{dens2} as a function of p, ϕ_{dens2}'s particle model is used. For this comparison, particle configurations at t_c in the CA's particle model are generated as follows. First, a total number n of particles is chosen from the empirically measured probability distribution $\Pr[n]$ as given in table 5.6. These n particles are then placed at random positions, with a uniform distribution over the lattice. Next, for each of the n particles it is decided whether it is of type α or of type β by tossing a biased coin with probability p for α and $1-p$ for β. Finally, t_c is set to the average t_c as measured for ϕ_{dens2}, which is $\overline{t_c} = 40$.

The third column in table 5.7 shows the performances of ϕ_{dens2} as a function of p as predicted by its particle model. These performances are averaged over 10 sets of 10,000 random initial particle configurations (generated as just explained). The standard deviations over these 10 sets are given in parentheses. As the table shows, the previously calculated performances are all well within one standard deviation from the corresponding model-predicted performances. Note that since the model-predicted performance of ϕ_{dens2} itself is 0.3377 (see table 5.2), the actual value of p for this CA will probably be close to 0.3 (section 6.2 returns to this issue of the actual value of p).

So, ϕ_{dens2}'s particle model can be used successfully to predict how changes in the PPD at t_c give rise to changes in performance. Furthermore, when a direct

p	$\mathcal{P}_{calc}$	$\mathcal{P}_{model}$
0.0	0.0302	0.0298 (0.0012)
0.1	0.1204	0.1218 (0.0027)
0.2	0.2298	0.2262 (0.0035)
0.3	0.3323	0.3325 (0.0042)
0.4	0.4047	0.4065 (0.0064)
0.5	0.4307	0.4308 (0.0051)

Table 5.7: Performance predictions for ϕ_{dens2} as a function of the probability p that a particle at t_c is of type α. $\mathcal{P}_{calc}$ gives the calculated performances using the empirically measured distribution $\Pr[n]$, and $\mathcal{P}_{model}$ gives the performances as predicted by ϕ_{dens2}'s particle model (standard deviations in parentheses).

approach for calculating the performances is possible, as in this case, the model results can be used to verify these direct calculations.

5.4.4 Case 4: ϕ_{parent} and ϕ_{child}

The final comparative analysis is done on two CAs that have not been analyzed here so far. These are CAs that occurred early on in a GA run on the global synchronization–1 task. One of these CAs, ϕ_{child}, is a direct descendent of the other CA, ϕ_{parent}, and differs in one bit position only. Despite this minimal difference in the lookup tables, there is a substantial difference in the particle logics of these CAs.

Both CAs have only one domain, namely the synchronized domain Λ^s. ϕ_{parent} has three particle types (α, β, and γ), but ϕ_{child} has only two particles (α and β). Furthermore, the β particle has changed velocity from 1/4 in ϕ_{parent} to 0 in ϕ_{child}. All particle interactions can have multiple outcomes, but the probabilities with which the different outcomes occur are different between the two CAs. The particle catalogs of both CAs are presented in appendix A. Figure 5.12 shows space-time diagrams of the two CAs (each with a different IC), with their particle-types labeled.

The performance of ϕ_{parent}, averaged over 10 sets of 10,000 ICs each, is 0.1826, while the performance of ϕ_{child}, measured over the same sets of ICs, is 0.2573. Using

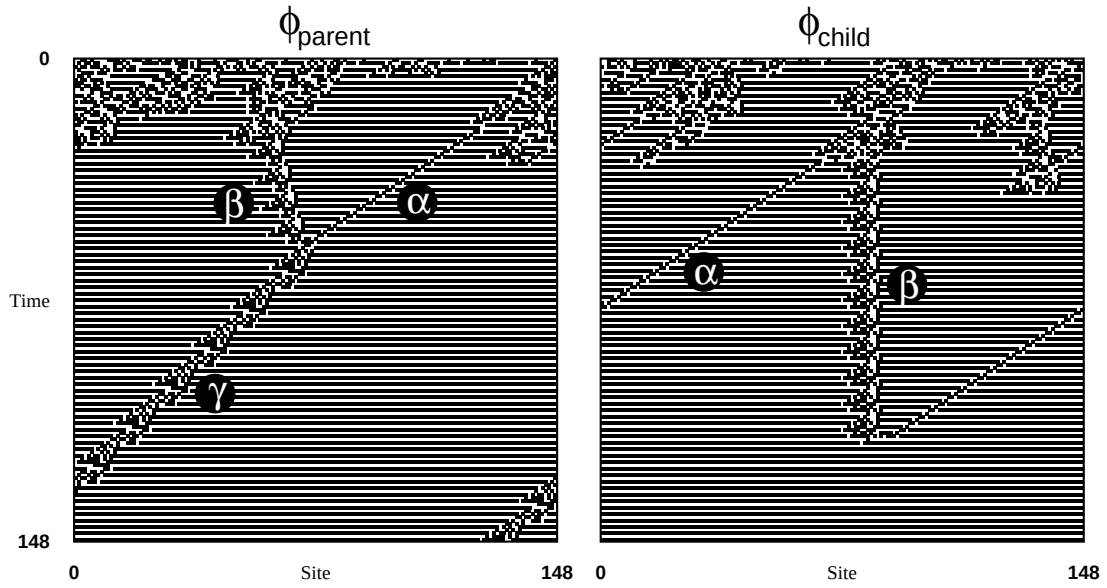

Figure 5.12: Space-time diagrams of ϕ_{parent} (left) and ϕ_{child} (right). The different particle types occurring in the CAs are labeled.

the particle models of these CAs, with initial particle configurations generated by running the CAs up to t_c, the predicted performances are 0.1858 and 0.2446, respectively. Table 5.8 summarizes these performance measurements. As with the other CAs that were evolved for the global synchronization–1 task, the performance predictions are within 5% of the CA performances.

	CA	Model	%
ϕ_{parent}	0.1826 (0.0034)	0.1858 (0.0036)	1.8
ϕ_{child}	0.2573 (0.0042)	0.2446 (0.0022)	4.9

Table 5.8: CA and model-predicted performances for ϕ_{parent} and ϕ_{child}.

The particle models can be used to investigate to what extent the differences between the particle logics of these two CAs contribute to the difference in performances. As in the other comparative analyses, one particular set of 10,000 random ICs is used to generate particle configurations at t_c as an approximation of the PPD

at t_c.

First, consider the velocity of the β particle. Using ϕ_{parent}'s particle model, but changing v_β from 1/4 to 0, induces no change in the predicted performance. Likewise, using ϕ_{child}'s model and changing v_β from 0 to 1/4 makes no difference. It can thus be concluded that the change in the velocity of the β particle is not a contributing factor to the difference in performances between the two CAs.

Next, consider the PPD at t_c. Using again ϕ_{parent}'s particle model, but the approximation of the PPD at t_c of ϕ_{child}, the predicted performance decreases from 0.1858 to 0.1649. Note that the PPD at t_c of ϕ_{child} does not include any γ particles. However, using ϕ_{parent}'s particle logic, γ particles can still be created after t_c, since one of the possible outcomes of an interaction between an α and a β particle, is a γ particle (with probability 0.3). The reverse, using ϕ_{parent}'s approximation of the PPD at t_c and the particle logic of ϕ_{child} is not possible, since the former includes γ particles, and the latter does not.

Now assume the creation of γ particles from the $\alpha + \beta$ interaction in ϕ_{parent}'s particle logic is eliminated. Note that this $\alpha + \beta$ interaction can have four different outcomes: annihilation with probability 0.3, a γ particle with probability 0.3, two α particles with probability 0.1, and another set of α and β particles with probability 0.3. In the particle catalog of ϕ_{parent}, the probability of the interaction $\alpha + \beta \rightarrow \gamma$ is now set to 0. This probability was 0.3, which is now added to the probability of the $\alpha + \beta \rightarrow \emptyset$ interaction, which thus becomes 0.6. Using ϕ_{parent}'s approximation of the PPD at t_c (which does include γ particles), the predicted performance now markedly increases to 0.3246. So, simply eliminating the possibility of creating a γ particle from the interaction of an α and a β particle (which normally occurs in about 30% of those interactions), almost doubles the performance.

Finally, combining both these changes (using the approximation of the PPD at t_c of ϕ_{child} and setting the probability of $\alpha + \beta \rightarrow \gamma$ to 0), gives a predicted performance of 0.2475, close to the predicted performance of ϕ_{child} itself. In other

words, using ϕ_{child}'s approximation of the PPD at t_c, now combined with the change in interaction probabilities, counteracts the increase in performance due to the change in particle interaction probabilities by itself.

Concluding, one single mutation between ϕ_{parent} and ϕ_{child} gives rise to several changes in the dynamics of these CAs, which result in a significant increase in performance. However, the different changes in the dynamics contribute in very different ways to this change in performance. The change in the velocity of the β particle does not contribute anything at all. Moreover, the change in the PPD at t_c actually decreases the CA's performance. However, the change in interaction result probabilities (in particular eliminating the $\alpha + \beta \rightarrow \gamma$ interaction) more than makes up for this, and causes the overall performance to increase significantly.

5.5 Conclusions

The analyses in this chapter have shown that the class of particle models is capable of accurately predicting the evolved CAs' computational performances. In many cases, the predictions are within 0.5% of the CA performances. In some cases, especially for the CAs evolved on the global synchronization–1 task, the discrepancies are larger, but are all still within 5%. Furthermore, these discrepancies can be attributed directly to the simplifying assumptions incorporated in the model class. So, the minor differences between a CA's dynamics and that as modeled by its particle model, and the subsequent discrepancies in CA and predicted performances, are well understood.

Next to predicting how often a CA settles down to a correct answer state, the class of particle models is also capable of predicting how long it takes a CA, on average, to settle down to an answer state. For some of these predictions, however, the discrepancies with the CA values are rather significant. But as with the performance predictions, these discrepancies can be explained directly by the simplifying assumptions in the models.

In previous work, it was claimed that a CA's particles and their interactions form the main mechanism by which the CA is capable of performing the global information processing necessary for accomplishing the given computational task. The generally close agreement between the performance predictions resulting from the class of particle models and the actual CA performances provides a direct verification of this claim. Furthermore, these results provide, for the first time, a direct and quantitative relation between a CA's dynamics and its (emergent) computational ability.

Beyond this relation between dynamics and computational ability, the class of particle models also provides a better understanding of the evolution of emergent computation in CAs. For example, the five CAs that were evolved on the density classification task are all related in an evolutionary sense, since they appeared in the same GA run, representing important innovations during their evolution. The performance predictions resulting from the particle models verify that these innovations are indeed due to differences in the computational strategies of these CAs. Similarly for the five global synchronization–1 CAs.

Furthermore, the class of particle models can be used to determine quantitatively to what extent these differences in the computational strategies of evolutionarily related CAs contribute to differences in these CAs' performances. For example, it was claimed previously that the difference between the performances of ϕ_{dens3} and ϕ_{dens4} is mainly due to the difference in the velocity of the β particle in these CAs. However, as the analysis with the particle models shows, this difference contributes only about 13% to the difference in performances. The main contribution turns out to be a difference in the PPD at t_c between these two CAs. Also, these differences contribute to the increase in performance in a nonlinear way. The increase in performance due to the combination of the two differences is more than the sum of the increases in performance due to each difference alone.

So, unlike in the CA itself, the particle models can be used to study the effects of certain properties of particles in isolation. In the CA, the properties of a particle,

like its velocity and the way it interacts with other particles, are inherently linked. It is generally impossible to change only one of these properties without affecting the others also. However, in a CA's particle model these properties are independent of each other, and can be studied in isolation.

In summary, the class of particle models, together with the results presented in this chapter, successfully addresses the four items stated as "still needed" in section 1.4 in the introduction:

1. The class of particle models formalizes the concept of an emergent computational strategy in a CA, in the form of a particle catalog plus an approximation of the PPD at t_c, and provides an algorithmic procedure for directly simulating a CA's computational strategy;

2. The particle models can be used to make quantitative predictions of the evolved CAs' computational performances;

3. The particle models are used to relate quantitatively changes in the computational strategy of a CA to changes in the CA's performance;

4. These changes are related to the evolutionary history of the evolved CAs by determining why and to what extent one CA's computational strategy is better than that of another, evolutionarily related, CA.

In conclusion, the class of particle models and the results it generates indeed provide a better and more formal understanding of the relation among dynamics, emergent computation, and evolution in cellular automata.

Chapter 6

Further Investigations of the Particle Models

The analyses in the previous chapter support the claim that the class of particle models captures the main mechanisms of the emergent computation that is evolved in the CAs. However, it could be argued that this support is indirect: it is obtained by quantitatively comparing CA and model performances and visually comparing space-time diagrams produced by a CA and its corresponding model. Thus, it does not provide a proof from first principles that a CA's particle model is correct.

This chapter provides this link. A proof is presented of the correctness of the particle model for a particular CA. It is shown that this particle model provides a complete description of the CA's dynamics. The CA is hand-constructed, but has a behavior similar to that observed during many GA runs. Its relatively simple behavior allows the proof to be worked out in a straightforward manner. For CAs with a more complex behavior, this type of proof will necessarily become more complicated. However, the proof presented here forms a basic framework for these more complicated proofs.

After the proof is completed, concise approximations of the PPD at t_c are derived for two evolved CAs. First, it is argued why it is difficult to find a general approximation that is both accurate and concise. An example of one concise general approximation is given, but it is shown that the results from this method are very inaccurate. Then, examples of concise approximations for two specific CAs are presented which do provide accurate results. Using these approximations, it is subsequently shown that it is possible to derive an expression for predicting these CAs' performance directly. In other words, the performance of these CAs can be predicted directly without actually running their particle models.

Finally, several scaling properties of the class of particle models are addressed. In particular, the scaling of the condensation time t_c and of the number of particles at t_c with lattice size N is investigated. Expressions for calculating the average condensation time $\overline{t_c}$ and for the number of particles at $\overline{t_c}$ as a function of lattice size N are derived for one particular evolved CA.

6.1 Correctness of a particle model

The proof presented in this section uses a simple hand-constructed CA. This CA, $\phi_{\text{bl-exp}}$, is a so called block expander, a computational strategy very often found early in GA runs on the density classification task. By default, this CA quickly settles down to an all-0s configuration, unless there is at least one block of at least five consecutive 1s in the IC. In that case, the CA expands these blocks, one cell per time step, until eventually the entire lattice becomes all 1s. The computational strategy of this CA is similar to that of ϕ_{dens2}. However, $\phi_{\text{bl-exp}}$ is explicitly constructed, and not evolved. Furthermore, $\phi_{\text{bl-exp}}$ expands blocks of black, while ϕ_{dens2} expands blocks of white, i.e., the two strategies are "mirror images".

The reason this hand-constructed CA $\phi_{\text{bl-exp}}$ is used here, and not ϕ_{dens2} for example, is that the emergent behavior of $\phi_{\text{bl-exp}}$, in terms of domains, particles, and particles interactions, does not violate any of the simplifying assumptions incorporated in the particle models. This facilitates a direct proof of the correctness of $\phi_{\text{bl-exp}}$'s particle model. Correctness proofs for other, more complex, CAs will follow the same basic framework as in the proof for $\phi_{\text{bl-exp}}$, but need to be adjusted for any emergent behavior that violates the simplifying assumptions.

Figure 6.1 shows two space-time diagrams of $\phi_{\text{bl-exp}}$ to illustrate its behavior. In the space-time diagram on the left, there are no blocks of 1s in the IC that are of length five or larger. In just a few time steps (two, in this case) the CA settles down to an all-0s configuration. In the space-time diagram on the right, however, there are two blocks of at least five consecutive 1s in the IC. The CA expands these blocks until eventually the lattice becomes all-1s (roughly around time step 60). Starting with the output bit for the all-0s neighborhood and assuming a lexicographical ordering of the local neighborhoods η, the lookup table of $\phi_{\text{bl-exp}}$ is as follows:

```
00000000000000010000000000000001000000000000000100000000000000011
00000000000000010000000000000001000000000000000111111111111111111
```

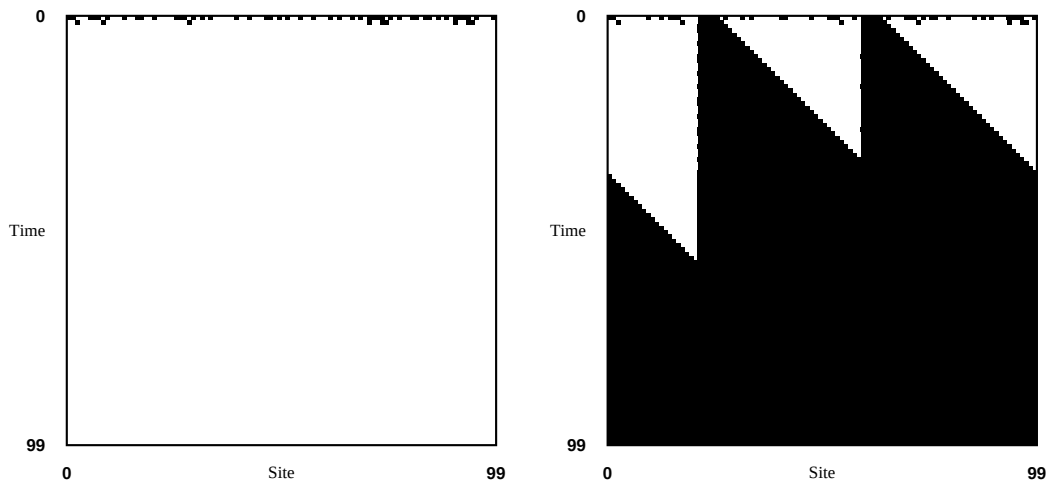

Figure 6.1: Space-time diagrams illustrating the behavior of $\phi_{\text{bl-exp}}$.

Judging from the space-time diagrams of $\phi_{\text{bl-exp}}$, there seem to be two candidate domains: the all-0s configuration 0^* and the all-1s configuration 1^*.

Lemma 1 $\Lambda^0 = 0^*$ *and* $\Lambda^1 = 1^*$ *are regular domains of* $\phi_{\textit{bl-exp}}$.

Proof. Both Λ^0 and Λ^1 are mapped onto themselves, since the output bit for the all-0s neighborhood is 0 (the first bit in the lookup table), and the output bit for the all-1s neighborhood is 1 (the last bit in the lookup table). Furthermore, the corresponding DFAs of Λ^0 and Λ^1 each have only one state and thus are strongly connected. So, $\Lambda^0 = 0^*$ and $\Lambda^1 = 1^*$ are both temporally invariant and spatially homogeneous and are thus regular domains of $\phi_{\text{bl-exp}}$. □

The boundaries between these domains form two particles, $\alpha = \Lambda^1\Lambda^0$ and $\beta = \Lambda^0\Lambda^1$. The periodicity of both these particles is $p = 1$. The displacement of the α particle is $d_\alpha = 1$; its velocity is $v_\alpha = 1$. The displacement of the β particle is $d_\beta = 0$; its velocity is $v_\beta = 0$. Using the expression given in section 2.4.4 for the number of possible particle interaction results, it turns out that the interaction between an α and a β particle can have at most $\frac{p_1 p_2 \Delta v}{p^t_\Lambda p^s_\Lambda} = \frac{1 \cdot 1 \cdot 1}{1 \cdot 1} = 1$ outcome. As the space-time diagram on the right in figure 6.1 shows, the α and β particles annihilate each other during an interaction. Concluding, the particle logic of $\phi_{\text{bl-exp}}$ appears to

be rather simple. This particle logic is summarized in the particle catalog shown in table 6.1.

Domains Λ				
Label	Regular language			
Λ^0	0^*			
Λ^1	1^*			
Particles P				
Label	Boundary	p	d	v
α	$\Lambda^1\Lambda^0$	1	1	1
β	$\Lambda^0\Lambda^1$	1	0	0
Interactions I				
$\alpha + \beta \rightarrow \emptyset$				

Table 6.1: The particle catalog of $\phi_{\text{bl-exp}}$.

$\phi_{\text{bl-exp}}$'s particle catalog, however, is primarily constructed by visual inspection of a small number of space-time diagrams. This does not guarantee that it completely captures $\phi_{\text{bl-exp}}$'s dynamics. However, it can be proved that this particle-level description indeed completely captures the dynamics of $\phi_{\text{bl-exp}}$. In other words, there is a one-to-one mapping between the space-time dynamics of $\phi_{\text{bl-exp}}$ and its particle model description and, furthermore, this description is complete in the sense that there does not exist any other CA behavior that is not captured in its particle model. The following lemmas provide the necessary steps for this proof.

Lemma 2 *For $\phi_{bl\text{-}exp}$ the condensation time $t_c \leq 3$.*

Proof. To prove this lemma, the FME algorithm (see section 2.3) is used. Recall that in this algorithm a finite state transducer T_ϕ representing a CA update rule ϕ is composed with a finite automaton M representing a set of lattice configurations. The minimal DFA M_{out} representing the output language of the transducer resulting from this composition is then obtained by minimizing with respect to the output symbols: $M_{out} = [T_\phi \circ M]_{out}$. This DFA then represents the set of possible lattice configurations

at the next time step, having started with the set M of configurations at the previous step.

The update transducer $T_{\phi_{\text{bl-exp}}}$ representing the update rule $\phi_{\text{bl-exp}}$ is given in appendix B. Next, consider the DFA M_{IC} that represents all possible ICs over the binary alphabet, i.e., $\{0,1\}^*$. This DFA is shown in figure 6.2.

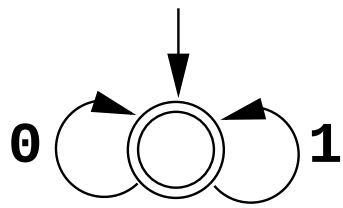

Figure 6.2: The DFA M_{IC} representing all possible ICs.

Using $T_{\phi_{\text{bl-exp}}}$ and M_{IC} in the FME algorithm, it can be shown[1] that the DFA $M_3 = [T^3_{\phi_{\text{bl-exp}}} \circ M_{IC}]_{out}$ which results after three iterations of the algorithm, is equal to the one shown in figure 6.3. This DFA represents all possible lattice configurations at time step $t = 3$.

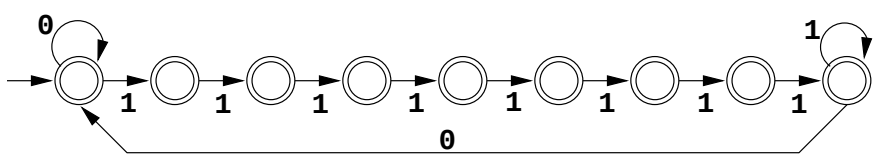

Figure 6.3: The DFA $M_3 = [T^3_{\phi_{\text{bl-exp}}} \circ M_{IC}]_{out}$ representing all possible lattice configurations at time step $t = 3$.

M_3 represents lattice configurations consisting of blocks of 1s of length eight or larger alternated by blocks of 0s or arbitrary length. In other words, regardless of the IC that the CA was started with, by time step $t = 3$ the lattice consist entirely of domains Λ^0 and Λ^1, with particles α and β in between them. Thus, over the set of all ICs $t_c \leq 3$. □

[1]Either by explicit construction or by using an automated version of the algorithm.

The next lemma states that a Λ^0 domain of length larger than six in between two Λ^1 domains decreases in size by one cell each time step.

Lemma 3 *The (local) lattice configurations* $1^*0^n1^*$ *are mapped to* $1^*0^{n-1}1^*$ *by* $\phi_{bl\text{-}exp}$, $n > 6$.

Proof. The proof of this lemma also uses the FME algorithm. For the update transducer $T_{\phi_{\text{bl-exp}}}$, again refer to appendix B. The DFA M of figure 6.4(a) represents the $1^*0^n1^*,\ n > 6$ configurations.

First, $T_{\phi_{\text{bl-exp}}}$ is composed with M. Figure 6.4(b) shows the resulting transducer. Next, the input symbols on the transitions of this resulting transducer are dropped. Figure 6.4(c) shows the resulting automaton. However, this automaton is nondeterministic, since there are two transitions with the label "1" coming out of state 0. Converting this NFA to an equivalent DFA results in the automaton shown in figure 6.4(d). Finally, this DFA is minimized, resulting in the DFA M' of figure 6.4(e). This DFA $M' = [T_{\phi_{\text{bl-exp}}} \circ M]_{out}$ represents configurations of the form $1^*0^{n-1}1^*, n > 6$, which proves the lemma. □

The next lemma proves that a Λ^1 domain of length larger than six in between two Λ^0 domains increases in size by one cell each time step.

Lemma 4 *The (local) lattice configurations* $0^*1^n0^*$ *are mapped to* $0^*1^{n+1}0^*$ *by* $\phi_{bl\text{-}exp}$, $n > 6$.

Proof. The proof is similar to that of the previous lemma, i.e., it follows by explicitly constructing $[T_{\phi_{\text{bl-exp}}} \circ M]_{out}$, where M is the DFA representing $0^*1^n0^*,\ n > 6$. The resulting DFA M' represents configurations of the form $0^*1^{n+1}0^*,\ n > 6$. □

Since lemma 2 shows that Λ^1 domains at t_c are always of length at least eight, the above lemma implies that *all* Λ^1 domains for $t \geq t_c$ increase in length by one cell each time step. The final lemma proves that a collision of an α and a β particle always leads to an annihilation. Combining this lemma with lemma 3 proves that *all* Λ^0 domains (of any length) decrease in length by one cell each time step for $t \geq t_c$.

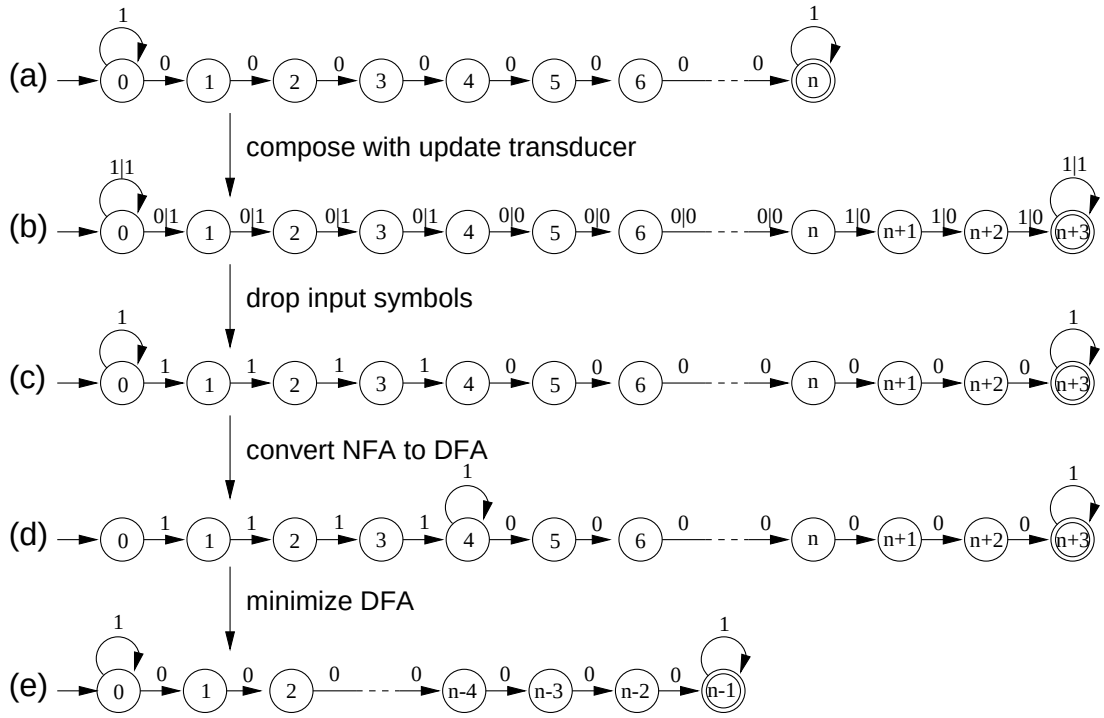

Figure 6.4: The steps in constructing $M' = [T_{\phi_{\text{bl-exp}}} \circ M]_{out}$.

Lemma 5 *Two particles α and β, starting n sites apart, move closer to each other by one cell each time step, and mutually annihilate upon collision n time steps later.*

Proof. Step 1. Two particles α and β n sites apart, $n > 6$, form a (local) lattice configuration $1^*0^n1^*$, $n > 6$. It follows directly from Lemma 3, then, that these two particles will be $n-1$ sites apart at the next time step. Thus, by induction, the first part of this lemma holds for $n > 6$.

Step 2. Using the FME algorithm iteratively, starting with $1^*0^61^*$, leads to the sequence of DFAs shown in figure 6.5. This sequence shows that the length of the Λ^0 domain decreases by one cell each time step, causing the α and β particles to move one cell closer at each time step. After six time steps, the Λ^0 domain and the α and β particles have completely disappeared, leaving Λ^1 occupying the lattice.

Combining steps 1 and 2 completes the proof. □

With the above lemmas proved, the next theorem follows straightforwardly.

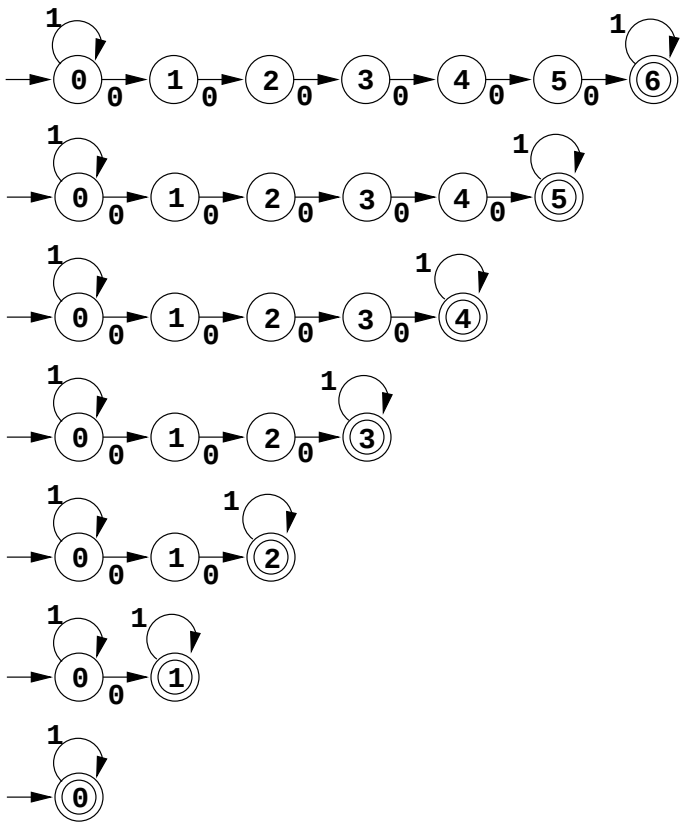

Figure 6.5: The sequence of DFAs resulting from applying the FME algorithm iteratively starting with the language $1^*0^61^*$.

Theorem 1 *The particle model of $\phi_{bl\text{-}exp}$ forms a complete description of its dynamics.*

Proof. In a CA's particle model, first a particle configuration at t_c needs to be constructed. This can be done by running the CA up to t_c. Lemma 2 shows that $t_c \leq 3$ for $\phi_{\text{bl-exp}}$, so to get a particle configuration at t_c, $\phi_{\text{bl-exp}}$ needs to be iterated for at most 3 time steps, regardless of the IC. Furthermore, lemma 2 shows that at t_c, $\phi_{\text{bl-exp}}$'s lattice consists entirely of Λ^0 domains of arbitrary length and Λ^1 domains of length at least eight, with α and β particles in between them.

Next, the above lemmas combined prove that for $\phi_{\text{bl-exp}}$ the Λ^0 domains decrease in length by one cell and the Λ^1 domains increase in length by one cell each time step for $t \geq t_c$. Since the α particle travels with velocity 1 and the β particle

with velocity 0, in $\phi_{\text{bl-exp}}$'s particle model a Λ^0 domain in between an α and a β particle decreases in length by one, and a Λ^1 domain in between a β and an α particle increases in length by one.

Finally, lemma 5 proves that an α and a β particle n sites apart move one cell closer at each time step, annihilating each other after n time steps. This, again, happens similarly in $\phi_{\text{bl-exp}}$'s particle model, since the interaction time t_i of two particles α and β n sites apart is calculated as $s + 1t_i = s + n + 0t_i \Rightarrow t_i = n$, where s is the location of the α particle and consequently $s + n$ is the location of the β particle.

In summary, lemma 2 proves that t_c is bounded—in fact, $t_c \leq 3$—and lemmas 3-5 combined prove that for $t \geq t_c$ the dynamics of $\phi_{\text{bl-exp}}$ can be completely described by the particle logic summarized in $\phi_{\text{bl-exp}}$'s particle catalog. In other words, no other configurations occur for $t \geq t_c$ than those that can be completely described in terms of the domains, particles, and particle interactions. □

The reason this proof works for $\phi_{\text{bl-exp}}$, as mentioned above, is that the dynamics of this CA does not violate the simplifying assumptions that are incorporated in the class of particle models. In particular, the particles α and β both have zero width, no more than two particles can interact at a time, the $\alpha + \beta$ particle interaction is instantaneous, and there is only one possible interaction result for this particle interaction. In this sense $\phi_{\text{bl-exp}}$ is an example of the most basic case. However, the above theorem proves that in this basic case the particle model is indeed complete and correct. In principle, the same basic framework can be used for more complex CAs.

6.2 Concise approximations of the PPD at t_c

In a CA's particle model, up to this point an approximation of the PPD at t_c has been generated by running the CA itself up to t_c and then using the actual particle

configuration at t_c as a starting point in the model. The accuracy of this approximation, of course, depends on the number of ICs on which this is done. The more ICs used, the more accurate the approximation is. However, it is not a very concise approximation of the PPD at t_c since it requires running the CA itself. It turns out that finding a general approximation that is both accurate *and* concise is a nontrivial problem.

There are a number of factors responsible for this. First, the periodic boundary conditions of the CA lattice need to be taken into account. That is, when using an approximation of the PPD to generate domain-particle configurations at t_c, there is the additional constraint that the generated configurations must wrap around correctly. In other words, the sites at the ends of the lattice need to be part of the same domain or particle. Second, there exist long range correlations in the CA configurations at t_c that extend beyond the direct neighbors of a cell. This occurs since, by t_c, information has been allowed to travel over a distance of $r \times t_c$ cells. These correlations must be captured in the PPD approximation for it to be accurate. Third, domains and particles are generated in parallel in the CA. Any approximation that is based on a sequential representation, e.g., some finite automaton that puts down particles while it traverses the lattice, is therefore likely to fail. Finally, the lengths of the domains at t_c (or equivalently, the distances between the particles) are important, since some of the evolved strategies more or less perform size competitions between the different domains. Whatever the approximation of the PPD is, the distribution of these domain lengths also must be reproduced accurately.

Several general solutions, i.e., solutions that would be applicable to any CA, have been tried. So far, none of them have produced sufficiently accurate results on all CAs. An example of such a failed attempt is given in the first subsection below, which helps to illustrate what makes an accurate general approximation of the PPD at t_c a difficult problem. However, in some cases it is possible to find a concise and accurate approximation of the PPD at t_c that is specific for one particular CA. Examples of

concisely and accurately modeling the PPD at t_c for two evolved CAs are given here. The first example is for a CA that is a block expander, but this time the CA is an evolved one. The second example is for ϕ_{sync2}.

6.2.1 A general but inaccurate approximation

One general method for approximating the PPD at t_c that was tried, is using the domain-particle-transducer. Recall that this transducer is used to determine the condensation time t_c in a CA's space-time diagram. The domain-particle-transducer recognizes the domains and particles of a CA, but raises a flag when an unrecognized spatial configuration is encountered while scanning the CA lattice.

The functionality of this domain-particle-transducer can be extended once more, by including a *transition probability matrix* $\mathcal{T}$. This transition matrix $\mathcal{T}$ will have one row (and one column) for each domain and each particle type that is recognized by the transducer. Initially the entries in $\mathcal{T}$ are set to 0. Eventually, these entries will contain the probabilities of making a transition from one recognized pattern (e.g., a domain) to another (e.g., a particle), while scanning a CA lattice at t_c.

This "transducer-plus-matrix" is now used as follows. The transducer scans a CA lattice as usual, where the lattice configuration at time t_c is used. Assume that the transducer starts scanning the lattice at a site that is currently contributing to some domain, say Λ^0. Suppose further that the next site that the transducer encounters (i.e., the next site in the lattice) is still part of the same domain Λ^0. In this case $\mathcal{T}[\Lambda^0, \Lambda^0]$ is increased by 1. Similarly, for every next site that the transducer encounters, while scanning the lattice, that is still part of the same domain Λ^0, $\mathcal{T}[\Lambda^0, \Lambda^0]$ is increased by 1.

At some point, the transducer might encounter a particle. Assume that this particle is of type α. As soon as the transducer has recognized the particle type (in this case α), $\mathcal{T}[\Lambda^0, \alpha]$ is increased by 1. The transducer then keeps scanning the

lattice until it reaches the other end of the particle (recall that particles can be several cells wide; the transducer might recognize the particle type only once it has scanned the entire width of the particle). So, during the steps that the transducer is scanning the particle, only once an entry in the transition matrix is updated, namely the entry corresponding to the transition from domain Λ^0 to particle α (in this example).

Once the particle is scanned completely, another domain, say Λ^1, will be encountered by the transducer. Consequently, $\mathcal{T}[\alpha, \Lambda^1]$ is increased by 1. Note that every time a particle of type α is encounter, a domain Λ^1 has to be encountered next, since a particle is defined by the domains between which is forms a boundary. Continuing scanning the lattice, $\mathcal{T}[\Lambda^1, \Lambda^1]$ is increased by 1 for each next cell that contributes to this domain Λ^1, until another particle is encountered. This updating of the appropriate entries in the transition matrix continues until the entire lattice is scanned by the transducer.

This process of scanning a CA lattice at t_c and updating the appropriate counts in the transition matrix $\mathcal{T}$ is then repeated on a large set of lattice configurations at t_c which result from starting the CA on random ICs. In this process, the transition counts made on one lattice configuration are "carried over" to the next. In other words, the transition counts over all lattice configurations are added into one transition matrix. Finally, these transition counts are converted into transition probabilities by dividing each entry $\mathcal{T}[i, j]$ in the matrix by the sum of the counts in the corresponding row i.

An additional probability distribution that is needed is the probability of being in a particular domain when starting to scan the lattice at an arbitrary site. This is easily obtained by counting, over the large set of lattice configurations at t_c, how often the transducer started in which domain. These counts are then converted into probabilities. This starting probability distribution s together with the transition probability matrix $\mathcal{T}$, then, are used as the approximation of the PPD at t_c for the particular CA being modeled. Note that s and $\mathcal{T}$ together define a Markov process.

As an example, consider ϕ_{dens5}. Recall that this CA has three domains (Λ^0, Λ^1, and $\Lambda^\#$) and five particle types (β, γ, δ, ε, and η). The particle catalog of ϕ_{dens5} is given in appendix A. Using the above transducer-plus-matrix method on a random sample of 10,000 ICs, results in the following vector of starting probabilities for the three domains

$$\mathbf{s} = (\Pr[\Lambda^0], \Pr[\Lambda^1], \Pr[\Lambda^\#]) = (0.4396, 0.3368, 0.2236)$$

and the following transition probability matrix $\mathcal{T}$ (empty entries denote a probability of 0)

	Λ^0	Λ^1	$\Lambda^\#$	β	γ	δ	ε	η
Λ^0	0.9663				0.0337			
Λ^1		0.9610		0.0329			0.0061	
$\Lambda^\#$			0.9278			0.0116		0.0606
β	1							
γ			1					
δ	1							
ε			1					
η		1						

This approximation of the PPD at t_c can now be used for generating particle configurations at t_c by actually iterating the Markov process defined by $\mathbf{s}$ and $\mathcal{T}$. First, a starting domain is chosen according to the starting probability distribution $\mathbf{s}$. Assume that domain Λ^0 is chosen to start with, which will happen with probability 0.4396 in the above example. The first site in the lattice will then be part of a domain Λ^0. Next, N (the lattice size) steps are made according to the transition probability matrix $\mathcal{T}$. In other words, the next site in the lattice will also be part of domain Λ^0 with probability $\mathcal{T}[\Lambda^0, \Lambda^0]$, which is 0.9663 in the example. And so on for every next site, until a transition to a particle is made.

Suppose at some point, say after i steps, a transition to a particle is made. In the example, this would be a particle of type γ, which is the only transition from Λ^0,

not returning to Λ^0, with a non-zero probability, 0.0337 in this case. Then, a particle of type γ is placed at site $i+1$. Next, a transition to another domain is made. According to the probability matrix this can only be $\Lambda^{\#}$. So now every next cell in the lattice is part of this domain $\Lambda^{\#}$, with probability 0.9278, until a transition to another particle is made (either a δ particle, with probability 0.0116, or a η particle, with probability 0.0606). This Markov process is repeated for N steps, i.e., until every site in the lattice is assigned to be either part of a domain or contains a particle of some type.

Note that it is possible that after creating a particle configuration this way, the lattice does not "wrap around" correctly. In other words, the domain or particle that the last site in the lattice is part of, does not necessarily agree with the domain that was started with. In this case, the particle configuration is discarded, and another one is generated, until one is obtained that obeys the periodic boundary condition constraint. Since the particle configurations are generated by a Markov process, it can be analytically calculated, using standard Markov chain analysis techniques [KS60, TK94], how often on average a particle configuration has to be generated before a valid one is obtained. Using the $\mathbf{s}$ and $\mathcal{T}$ from the above example, this comes out to be about 2.9 times on average.

To check the accuracy of this approximation of the PPD at t_c, figure 6.6 shows a comparison of the actual relative frequencies of the total number of particles occurring at t_c (bars) for ϕ_{dens5} with the relative frequencies that result from the Markov chain procedure (dots connected by solid line), using $\mathbf{s}$ and $\mathcal{T}$ from the example above. As the figure shows, there is a large difference between the actual PPD at t_c and the approximation. Both histograms were generated from 10,000 particle configurations. As it turns out, to get 10,000 valid particle configurations from the Markov chain procedure, 28,106 trials were necessary. This is indeed very close to the calculated average of 2.9 trials to get one valid particle configuration.

Concluding, the current method for approximating the PPD at t_c and gen-

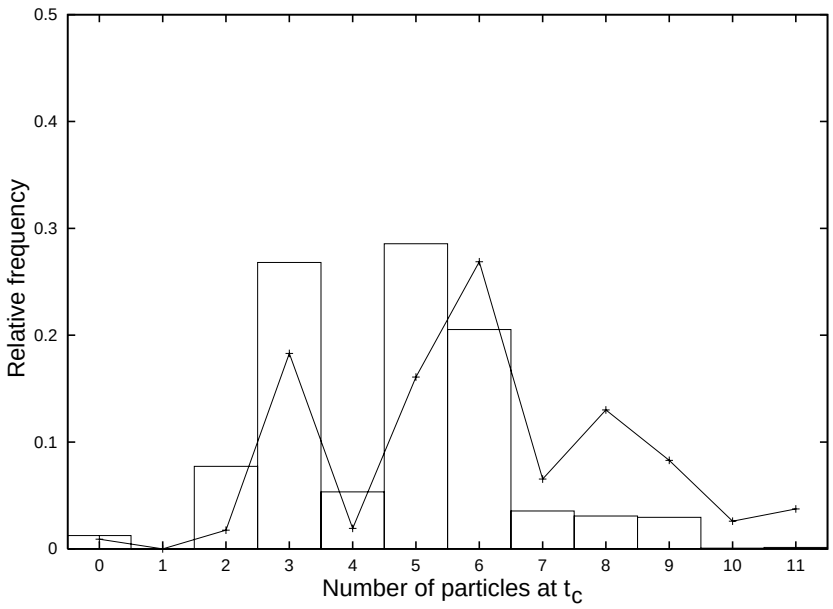

Figure 6.6: Relative frequencies of total number of particles at t_c for ϕ_{dens5} (bars) and as generated by the approximation of the PPD at t_c (solid line).

erating particle configurations does not produce accurate results. There is a large difference between the actual and the approximated PPD at t_c. There are several reasons why this approach does not work well. First, it creates the particle configurations at t_c sequentially, while in the actual CA they are generated in parallel. Second, the boundary conditions are not taken into account, at least not in a direct way. Third, the Markov process induces an exponential probability distribution for the lengths of the domains in the particle configurations that are generated. This is generally not the case in the evolved CAs. In other words, there are too many differences between the actual CA and the Markov process to make it work well.

Other general purpose methods that were tried did not produce any better results than the above Markov process method. There are several global constraints that need to be taken into account (such as the ones just mentioned), and it is difficult to capture all of those in a general method, i.e., one that would work well for all (evolved) CAs. However, for certain specific CAs, not all these global constraints apply, and sometimes it is possible to find a CA-specific approximation of the PPD at

t_c that does produce very accurate results. The next two subsections give examples of such cases.

6.2.2 An accurate approximation for the evolved block expander

The CA used in this first example, $\phi_{\text{ev-be}}$, appeared during a GA run on the density classification task. Its behavior is similar to that of $\phi_{\text{bl-exp}}$, the hand-constructed block expander. However, $\phi_{\text{ev-be}}$ expands blocks of six or more consecutive black cells in the IC, whereas $\phi_{\text{bl-exp}}$ expands block of five or more black cells. Furthermore, the average condensation time for $\phi_{\text{ev-be}}$ is larger than that of $\phi_{\text{bl-exp}}$, and occasionally it actually creates a block of six or more black cells during the condensation phase, which it then expands. Thus, its behavior during the condensation phase is more complicated than that of $\phi_{\text{bl-exp}}$. Figure 6.7 shows space-time diagrams of the evolved block expander $\phi_{\text{ev-be}}$.

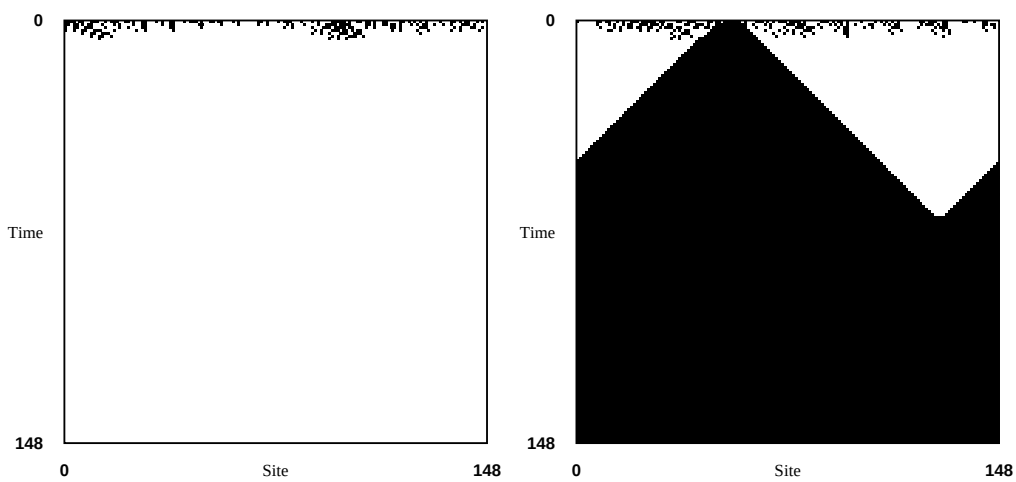

Figure 6.7: Space-time diagrams illustrating the behavior of $\phi_{\text{ev-be}}$.

$\phi_{\text{ev-be}}$ has two domains: the white domain $\Lambda^0 = 0^*$ and the black domain $\Lambda^1 = 1^*$. There are two particles, $\alpha = \Lambda^1\Lambda^0$ and $\beta = \Lambda^0\Lambda^1$, which annihilate each other upon collision. The particle velocities are $v_\alpha = 1$ and $v_\beta = -1$. The particle

logic of $\phi_{\text{ev-be}}$ is summarized in table 6.2. For this CA, the average condensation time is measered to be $\overline{t_c} = 8$.

Domains Λ				
Label	Regular language			
Λ^0	0^*			
Λ^1	1^*			
Particles P				
Label	Boundary	p	d	v
α	$\Lambda^1\Lambda^0$	1	1	1
β	$\Lambda^0\Lambda^1$	1	-1	-1
Interactions I				
$\alpha + \beta \rightarrow \emptyset$				

Table 6.2: The particle catalog of $\phi_{\text{ev-be}}$.

In order to model the PPD at t_c of $\phi_{\text{ev-be}}$ accurately, the following observations are helpful:

1. Particles at t_c always occur in pairs of an α and a β particle which form the right and left boundaries, respectively, of a Λ^1 domain.

2. Λ^1 domains occur only when there is a block of six or more consecutive black cells in the IC (or when such a block is created during the condensation phase).

3. Λ^1 domains and Λ^0 domains alternate each other in the lattice. If there are no Λ^1 domains at t_c, the entire lattice consists of one Λ^0 domain.

From these observations, it follows that a lattice configuration at t_c can be represented completely by specifying the total number, the lengths, and the positions of the Λ^1 domains. So, for modeling the PPD at t_c, three probability distributions are needed, one each for: (1) the total number of Λ^1 domains at t_c; (2) the lengths of Λ^1 domains at t_c; and (3) the positions of the Λ^1 domains at t_c.

The first two distributions can easily be obtained empirically. The probability distribution $\Pr[n]$ of the total number n of Λ^1 domains at t_c, measured over a set of

10,000 random ICs of length $N = 149$, is shown in table 6.3. Note that the measured probabilities are split into two distributions, one measured over 5,000 random ICs with $\rho_0 < 0.5$ and one measured over 5,000 random ICs with $\rho_0 > 0.5$. This is necessary later on when these empirical distributions are used in $\phi_{\text{ev-be}}$'s particle model to predict the CA's performance, as will be explained below.

	$\Pr[n]$	
n	$\rho_0 < 0.5$	$\rho_0 > 0.5$
0	0.3255	0.0957
1	0.4774	0.4458
2	0.1770	0.3767
3	0.0197	0.0767
4	0.0004	0.0049
5	0.0000	0.0002
≥ 6	0.0000	0.0000
$\overline{n}$	0.89	1.45

Table 6.3: Empirically measured probability distribution $\Pr[n]$ of the total number n of Λ^1 domains at t_c.

The frequency distribution of the lengths l of Λ^1 domains at t_c is shown in figure 6.8, again split into distributions over ICs with $\rho_0 < 0.5$ and ICs with $\rho_0 > 0.5$. As the plot shows, the distribution is shifted more to the right for $\rho_0 > 0.5$, compared to the one for $\rho_0 < 0.5$. This is no surprise, since one can expect more blocks of 6 or more black cells in ICs with a high density than in ICs with a low density. These frequency distributions can be easily converted into their corresponding probability distributions $\Pr[l]$.

For the third probability distribution $\Pr[s]$ of the positions s of the Λ^1 domains, a uniform distribution over the lattice is assumed, i.e., $\Pr[s] = 1/N$. The reason behind this is that a block of six or more black cells can occur anywhere in the IC, with equal probability. Consequently, the positions of Λ^1 domains also have a uniform distribution over the lattice. This follows from observation 2 above. Uniformity is furthermore supported by empirical measurements (not shown).

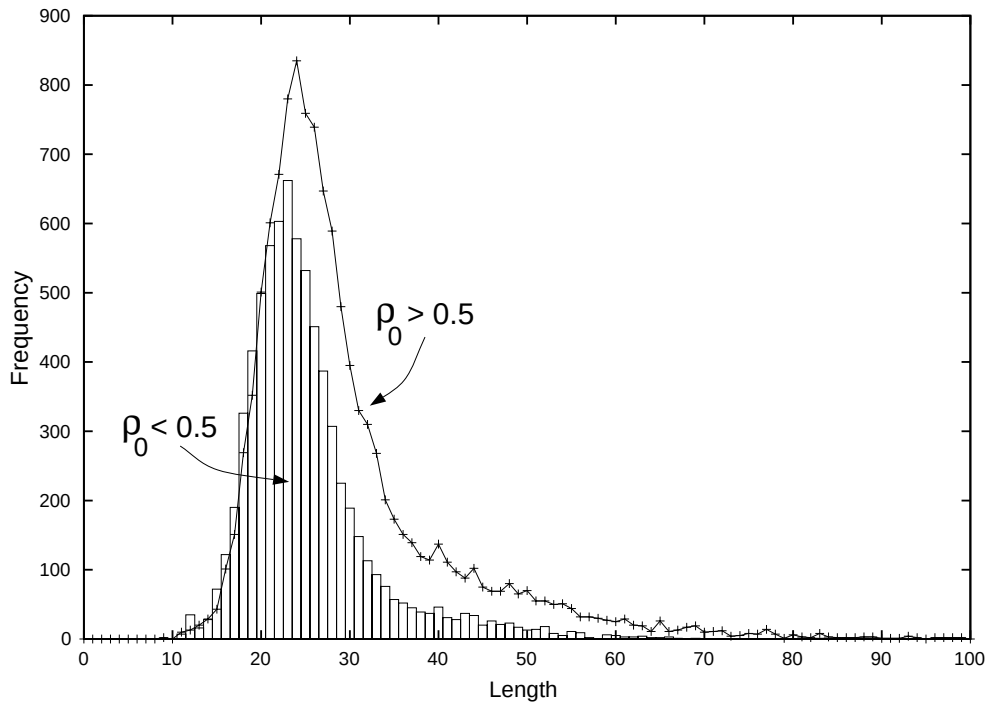

Figure 6.8: Frequency distributions of the length l of a Λ^1 domain at t_c. Bars indicate the distribution measured over 5,000 ICs with $\rho_0 < 0.5$, and +'s connected by the solid line indicate the distribution measured over 5,000 ICs with $\rho_0 > 0.5$.

Using these three probability distributions, the PPD at t_c of $\phi_{\text{ev-be}}$ is modeled as follows. First, a total number n of Λ^1 domains is drawn from the first probability distribution $\Pr[n]$. Next, for each of the n Λ^1 domains a length l is drawn independently from the second probability distribution $\Pr[l]$. Finally, the n Λ^1 domains are placed at random positions s in the lattice, according to a uniform distribution. Of course, when placing the Λ^1 domains in the lattice, overlapping domains must be avoided. Once the n Λ^1 domains are placed in the lattice, the types and locations of the particles are known too, since there is an α particle to the right of each Λ^1 domain and a β particle to the left of each Λ^1 domain. This, then, constitutes a particle configuration at t_c, where $\overline{t_c} = 8$ is used as the value for t_c in the particle model.

Figure 6.9 shows a comparison of the frequency distribution of the total number of particles at t_c for $\phi_{\text{ev-be}}$ and of that generated by the above model of the PPD at

t_c. The CA results are measured over 5,000 ICs with $\rho_0 < 0.5$. The model results are generated by creating 5,000 initial particle configurations using the probability distributions $\Pr[n]$ and $\Pr[l]$ that were measured over the ICs with $\rho_0 < 0.5$. The bars in the plot show the frequencies as measured in the CA and the dots connected by a solid line show the frequencies resulting from the PPD model. Note that the probability of having an odd number of particles at t_c is zero, since particles always have to occur in pairs.

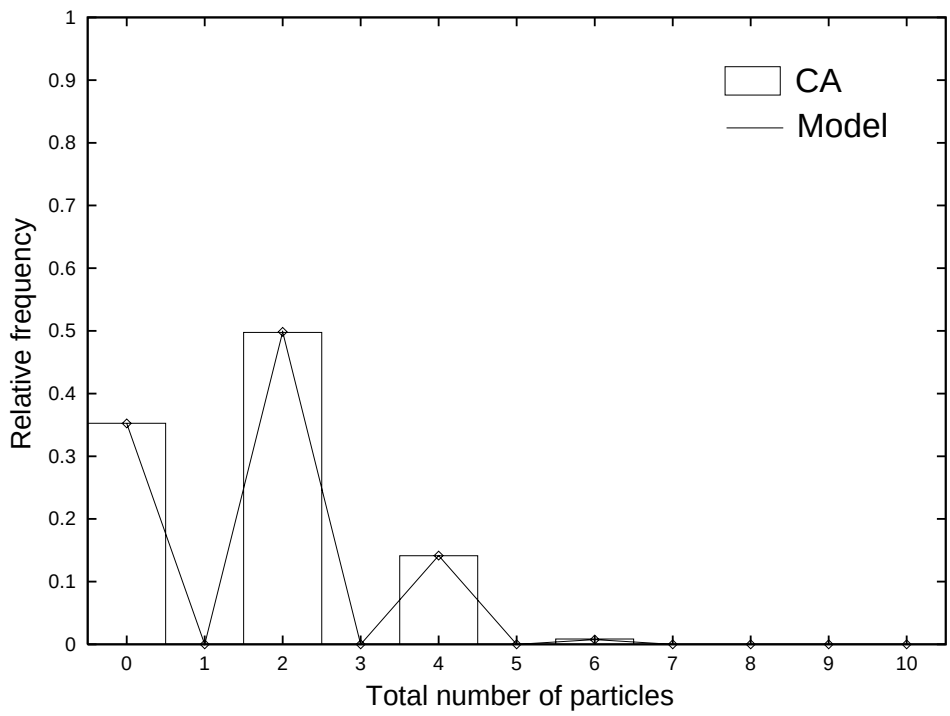

Figure 6.9: A comparison of the relative frequencies of the measured number of particles at t_c (`CA`) and those generated by the PPD model (`Model`).

As the figure shows, the agreement is excellent. A χ^2-test comparing the two distributions yields a p-value of 0.8. Using a significance level of 0.05, this means that there is no reason to reject the hypothesis that the two distributions are the same, since $0.8 \gg 0.05$. The results generated with the set of ICs with $\rho_0 > 0.5$ are similar.

$\phi_{\text{ev-be}}$'s performance can now be predicted using the PPD model as follows. First, an initial particle configuration at t_c is generated from the PPD model as

explained above. If the probability distributions $\Pr[n]$ and $\Pr[l]$ measured over the ICs with $\rho_0 < 0.5$ were used in generating the initial particle configuration, then the correct answer state is the white domain. If the probability distributions measured over ICs with $\rho_0 > 0.5$ were used, then the correct answer state is the black domain. Now, with initial particle configurations generated this way, the CA's particle model is run as before, and it is checked whether the simulated dynamics settles down to the correct answer state within M time steps. This is repeated a large number of times, e.g., 5,000 times with the $\rho_0 < 0.5$ distributions and 5,000 times with the $\rho_0 > 0.5$ distributions. The predicted performance is then simply the fraction of times the correct answer state is reached, just as before.

The actual performance of $\phi_{\text{ev-be}}$, averaged over 10 measurements on 10,000 random ICs each (5,000 with $\rho_0 < 0.5$ and 5,000 with $\rho_0 > 0.5$) is 0.6133, with a standard deviation of 0.0045. The predicted performance, using the above PPD model as just explained, also averaged over 10 measurements, is 0.6143, with a standard deviation of 0.0033. Clearly, these performance measurements agree well within one standard deviation.

As the results show, the PPD model approximates the PPD at t_c accurately and can be used in the particle model to also predict the CA's performance accurately. Furthermore, in addition to being accurate, the PPD model is very concise as it uses only three simple probability distributions instead of running the CA.

6.2.3 An accurate approximation for ϕ_{sync2}

In a similar way, using empirical probability distributions, the PPD at t_c of ϕ_{sync2} can be accurately and concisely modeled. Figure 3.4 shows a space-time diagram of this CA. Its particle catalog is presented in appendix A. ϕ_{sync2} has one domain Λ^s (the synchronized pattern) and two particles, α and β, which annihilate each other upon collision. The average condensation time is $\overline{t_c} = 40$.

ϕ_{sync2} was used in section 5.4, where its performance was calculated as a

function of the probability p that a particle existing at t_c is of type α. It was then argued that the actual value of p for this CA is close to 0.3. However, as shown here, the value of p is dependent on the total number n of particles at t_c.

To model the PPD at t_c for ϕ_{sync2}, three probability distributions are required, one each for: (1) the total number n of particles at t_c; (2) the number n_α of type α particles, given n total particles at t_c; and (3) the positions of the particles at t_c. As with $\phi_{\text{ev-be}}$, the first two distributions are measured empirically and the third one is assumed to be uniform over the lattice.

The probabilities $\Pr[n]$ of the total number n of particles at t_c, as measured over a set of 10,000 random ICs of length $N = 149$, is presented in table 6.4. The conditional probabilities $\Pr[n_\alpha|n]$ of the number n_α of type α particles, given n total particles at t_c, measured over the same 10,000 random ICs, is presented in table 6.5. As this table shows, the probability that a particle at t_c is of type α depends on the total number n of particles at t_c. Finally, the assumption of particles being uniformly distributed over the lattice is again confirmed by empirical measurements (not shown).

Note that for ϕ_{sync2} it is not necessary to split the measured probabilities into two distributions, since for the global synchronization tasks there is no relevant information about the IC that needs to be stored. There is only one correct answer state, the globally synchronized state, regardless of the IC.

Using these three probability distributions, the PPD at t_c of ϕ_{sync2} is modeled as follows. First, a total number n of particles is drawn from the first probability distribution $\Pr[n]$. Given n, a number n_α is drawn from the second, conditional, distribution $\Pr[n_\alpha|n]$. Then, n_α particles are assigned type α and the remaining $n - n_\alpha$ particles are assigned type β, at random. Finally, the n particles are placed in the lattice with a uniform distribution over the lattice, avoiding overlaps. This then constitutes an initial particle configuration at t_c, where $\overline{t_c} = 40$ is used as the value for t_c.

n	$\Pr[n]$
0	0.0302
1	0.0
2	0.3621
3	0.0
4	0.4769
5	0.0
6	0.1235
7	0.0
8	0.0073
≥ 9	0.0
$\overline{n}$	3.43

Table 6.4: The empirically measured probability distribution $\Pr[n]$ of the total number n of particles at t_c.

		$\Pr[n_\alpha \mid n]$								
		0	1	2	3	4	5	6	7	8
	0	1								
	2	0.4888	0.4060	0.1052						
n	4	0.2218	0.3454	0.2795	0.1285	0.2474				
	6	0.0883	0.1862	0.2502	0.2615	0.1425	0.0640	0.0073		
	8	0.0137	0.0411	0.0822	0.1918	0.2192	0.3288	0.0685	0.0547	0.0000

Table 6.5: The empirically measured conditional probabilities $\Pr[n_\alpha|n]$ of the number n_α of type α particles given a total number n of particles at t_c.

Figure 6.10 shows a comparison of the frequency distributions of the number of particles at t_c for both particle types for ϕ_{sync2} and of those generated by the above PPD model, measured over 10,000 random ICs. The bars show the actual frequencies (labeled "CA"), and the dots connected by a solid line show the numbers generated by the PPD model (label "Model"). The top plot is for the α particle and the bottom plot for the β particle. As the figure shows, the agreement is excellent. A χ^2-test yields p-values of 0.6 and 0.45 for the respective particle types. In other words, using a significance level of 0.05, there is no reason to reject the hypothesis that the CA

and model distributions are the same.

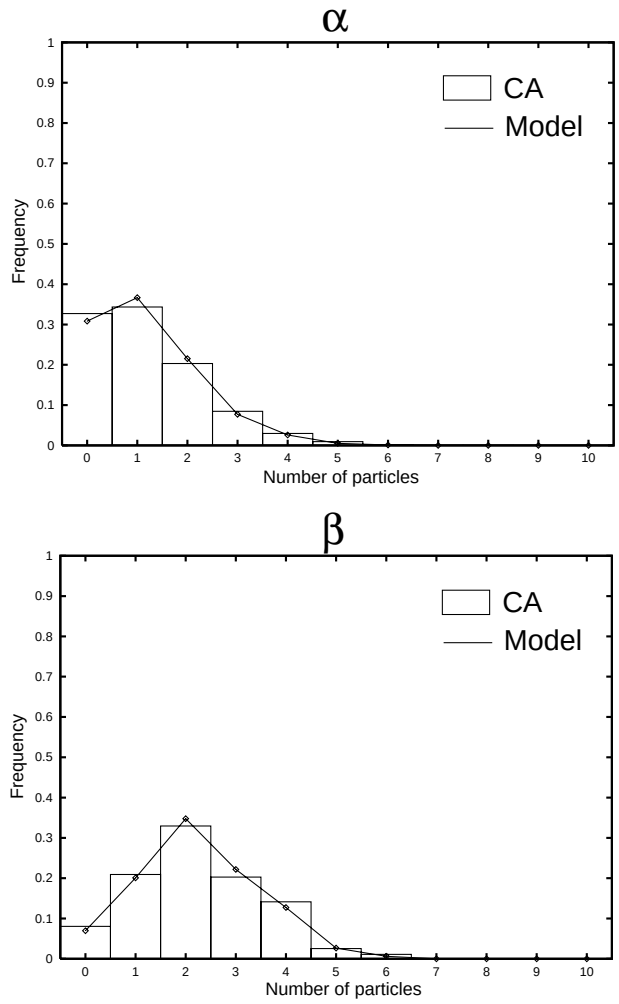

Figure 6.10: A comparison of the number of particles at t_c (`CA`) with the numbers generated by the PPD model (`Model`). The top plot shows the results for the number of α particles at t_c, and the bottom plot shows the results for the number of β particles.

The performance of ϕ_{sync2}, averaged over 10 measurements on 10,000 random ICs each, is 0.3198 with a standard deviation of 0.0054. Using the above PPD model to generate initial particle configurations at t_c, the performance predicted by the CA's particle model is 0.3173 with a standard deviation of 0.0035. Again, the performance prediction using the PPD model is well within one standard deviation of the CA performance.

Concluding, the above two examples show that, at least in some cases, it is possible to find an accurate and concise (using just a few simple probability distributions) approximation of the PPD at t_c, which can then be used in a CA's particle model to generate particle configurations at t_c. However, such an approximation is applicable only to the particular CA for which it was derived. In other, more complicated, cases (e.g., when there are more domains and particle types) the same modeling framework of using empirical probability distributions can still be used, but such an approximation would become rather complicated in practice, for example requiring several levels of conditional probabilities. Thus, finding an accurate and concise method of approximating a CA's PPD at t_c that is generally applicable to all CAs still remains an open problem.

6.3 Direct performance calculations

In section 4.5.2 it was shown that the computational time complexity of a CA's particle model is lower than that of the CA itself. In other words, it takes less time to run the particle model than it takes to run the CA. In terms of predicting a CA's performance, in some cases it is not even necessary to run the CA's particle model, since an expression can be derived for calculating this prediction directly. However, this requires an accurate and concise approximation of the PPD at t_c. In this section, expressions for calculating directly the predicted performances of the two CAs from the examples in the previous section are derived, and the calculated values are compared to those resulting from the particle models.

6.3.1 The evolved block expander

For $\phi_{\text{ev-be}}$, the correct answer state depends on the density ρ_0 of the IC. If $\rho_0 < 0.5$ the correct answer is a Λ^0 domain that occupies the entire lattice. In case $\rho_0 > 0.5$, the correct answer is a Λ^1 domain. Furthermore, from the behavior of $\phi_{\text{ev-be}}$, it is

known that if there is at least one Λ^1 domain present at t_c, then the entire lattice will eventually become all 1s; otherwise, it will become all 0s.

Consequently, the probability of a correct answer on a low-density IC ($\rho_0 < 0.5$) is the probability that there are no Λ^1 domains at t_c. Denote this probability by $\Pr[\text{no } \Lambda^1 |\ \rho_0 < 0.5]$. Similarly, the probability of a correct answer on a high-density IC ($\rho_0 > 0.5$) is the probability that there is at least one Λ^1 domain at t_c. Denote this probability by $\Pr[\geq 1\ \Lambda^1 |\ \rho_0 > 0.5] = 1 - \Pr[\text{no } \Lambda^1 |\ \rho_0 > 0.5]$. Now, the predicted performance $\mathcal{P}$ of $\phi_{\text{ev-be}}$ is simply given by

$$\mathcal{P} = 1/2 \left[\Pr[\text{no } \Lambda^1 |\ \rho_0 < 0.5] + (1 - \Pr[\text{no } \Lambda^1 |\ \rho_0 > 0.5])\right]$$

That is, $\mathcal{P}$ is the average of the probabilities of a correct answer on low-density ICs and a correct answer on high-density ICs.

Table 6.3 in the previous section presented the empirical probabilities $\Pr[n]$ of the total number n of Λ^1 domains at t_c. This was split into two distributions, one for ICs with $\rho_0 < 0.5$ and one for ICs with $\rho_0 > 0.5$. Using the probabilities from that table in the expression above, the predicted performance of the evolved block expander is calculated directly as

$$\mathcal{P} = \frac{0.3255 + (1 - 0.0957)}{2} = 0.6149$$

The predicted performance obtained from running the CA's particle model, as given in the previous section, is 0.6143 with a standard deviation of 0.0033. Thus, the directly calculated predicted performance is well within one standard deviation of the model-predicted performance.

Note that this direct calculation of the performance of $\phi_{\text{ev-be}}$ is possible because the answer state that this CA settles down to depends solely on the presence or absence of black domains at t_c. However, this reasoning cannot be extended to the analysis of general CAs, since this property is not common to all CAs. In fact, it is only a property of a restricted class of CAs, namely those that behave as block expanders.

6.3.2 ϕ_{sync2}

For ϕ_{sync2}, the correct answer of the globally synchronized state is reached only when there is an equal number of α and β particles at t_c (see the discussion in section 5.4). Denote the probability for the total number n of particles at t_c by $\Pr[n]$ and the number n_α of type α particles given n total particles at t_c by $\Pr[n_\alpha|n]$, as before. Given these probabilities, the predicted performance $\mathcal{P}$ of ϕ_{sync2} is then calculated directly as

$$\mathcal{P} = \sum_{n \text{ even}} \Pr[n] \times \Pr[n_\alpha = n/2|n]$$

That is, $\mathcal{P}$ is the sum over all even values of n of the probability of having n particles at t_c times the probability that half of those are of type α.

The empirically measured probabilities $\Pr[n]$ and $\Pr[n_\alpha|n]$ were given in tables 6.4 and 6.5, respectively. Using these empirical measurements, the above exression evaluates to $\mathcal{P} = 0.3444$. The predicted performance obtained from running ϕ_{sync2}'s particle model, however, is 0.3198, with a standard deviation of 0.0054. So, these two performance predictions do not match very well.

In the direct calculation of the performance prediction it is assumed that when there are equal numbers of α and β particles at t_c, then they all annihilate within the maximum number of times steps M. However, in the CA (or in its particle model, for that matter), this turns out to not always be the case. Sometimes an α and a β particle do not have a sufficient amount of time left after t_c to collide and annihilate. Since $\overline{t_c} = 40$ for ϕ_{sync2}, on average, a pair of α and β particles has roughly 260 time steps remaining to annihilate ($M = 2N = 2 \times 149 = 298$). However, an α and a β particle move closer to each other only with a relative velocity of 1/2. Consequently, when they are more than $260/2 = 130$ cells apart at t_c, they will not collide before time step 298. And since the lattice is of length $N = 149 > 130$, such a particle configuration is certainly possible.

To account for this difference, the maximum number of times steps M in

ϕ_{sync2}'s particle model is therefore set to $M = 400$, to give all particles at t_c enough time to eventually annihilate each other. Calculating the predicted performance again with the particle model with this new value for M, gives a result of 0.3442 with a standard deviation of 0.0043. Clearly, the agreement between this value and the directly calculated predicted performance of 0.3444 is excellent.

Concluding, the examples in this section show that in some specific cases it is possible to find an expression to calculate directly the predicted performance of a CA. This type of approximation eliminates the need to run a CA's particle model to predict its performance. However, the existence of such expressions depends crucially on the existence of a concise and accurate approximation of the PPD at t_c.

6.4 Lattice size scaling

One of the general benefits of the particle model of a CA is that the particle catalog is independent of the lattice size on which the CA is run. This catalog contains information about local structures that can occur anywhere in the lattice, regardless of the lattice size. The second part of a particle model, the approximation of the PPD at t_c, however, does depend on the lattice size. For example, when the CA is run up to t_c on a lattice of size N, the particle configuration at t_c is only valid for that particular lattice size N. Furthermore, in the approximations of the PPD at t_c used in the previous sections, the probability distributions were measured on one particular lattice size only. It is not directly clear how they would generalize to arbitrary lattice sizes. Finally, when $\overline{t_c}$ is measured for a CA, it is also valid only for the particular lattice size on which it was measured.

In this section, these lattice size scaling issues are addressed. Using the evolved block expander $\phi_{\text{ev-be}}$, it is shown that an expression can be derived for the expected values for t_c and for the total number n of particles at t_c as a function of the lattice size. For this derivation, first a particular (relatively small) lattice size N is taken as

a base case. The empirical probability distributions for t_c and n for this base case are then measured. Then, for larger lattice sizes $k \times N$, it is assumed that this larger lattice consists of k independent sublattices of size N. Finally, using the theory of order statistics (explained below), the expected values of t_c and n for the larger lattice sizes $k \times N,\ k = 2, 3, \ldots$, are calculated directly.

Figure 6.11 illustrates this approach graphically. In the top figure, a schematic (filtered) space-time diagram with lattice size N is shown for $\phi_{\text{ev-be}}$. The condensation time is indicated by the solid horizontal line labeled t_c. There are $n = 2$ particles at t_c, which annihilate each other upon collision. The bottom figure shows a space-time diagram with lattice size $4N$ (i.e., $k = 4$). It is divided into 4 sublattices of size N by the vertical dashed lines. The horizontal dashed lines indicate the condensation times in each sublattice if it were considered in isolation. The solid horizontal line indicates the condensation time for the entire lattice of size $4N$. This condensation time, of course, coincides with the condensation time of the sublattice that happens to condense last, which in this example is sublattice 3. Given the (empirical) probability distributions for t_c and n on lattice size N, the expected values for t_c and n on the lattice size $4N$, or any lattice size kN for that matter, can then be calculated using order statistics.

The theory of *order statistics* [Dav81, BE87] is a useful tool when, given a sample of a certain size k, one is interested in the statistics of the *ordered* set of observed values. Consider the above example of the lattice of size $4N$ consisting of four sublattices of size N. Suppose that the respective t_c values of these four sublattices are 4, 17, 20, and 9. To find the condensation time of the entire lattice of size $4N$, it is not relevant that is was sublattice 3 that condensed last. It might as well have been any of the other sublattices. What is relevant here, is the particular value of the largest t_c value among the four sublattices. The theory of order statistics provides, among others, a means for deriving the probability distribution of this largest t_c value, regardless of in which sublattice it occurs, given the probability distribution

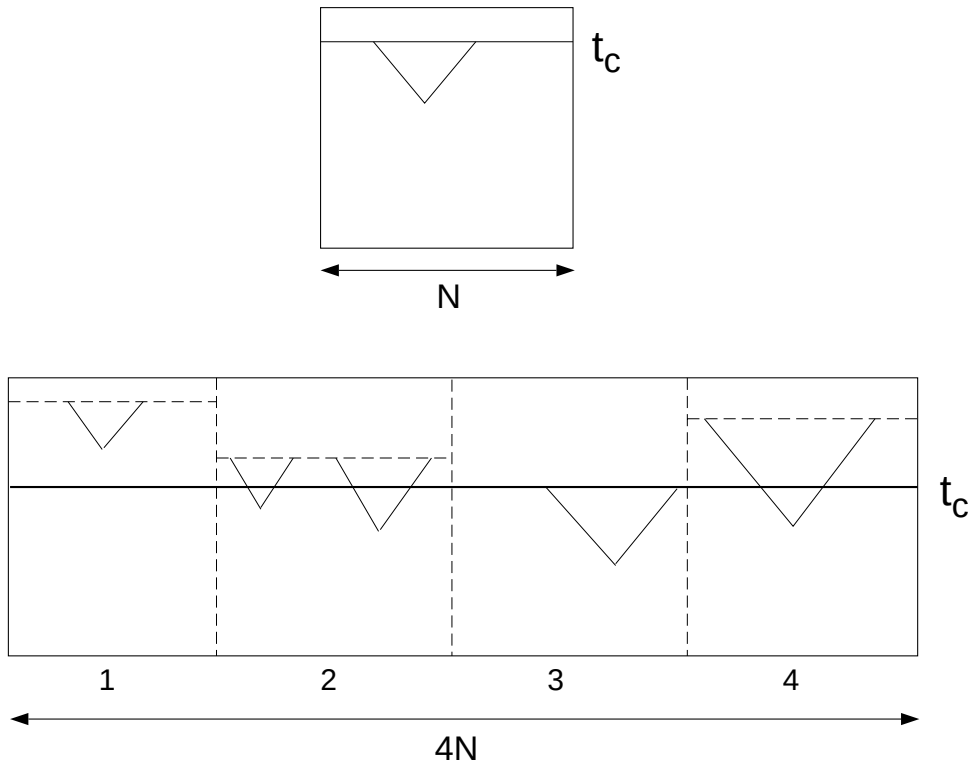

Figure 6.11: Top figure: A schematic (filtered) space-time diagram on lattice size N. Bottom figure: A schematic (filtered) space-time on lattice size $4N$, divided into 4 sublattices of size N. The condensation time of the entire lattice of size $4N$ coincides with the condensation time of the sublattice that condenses last.

of t_c values on the base case lattice of size N. More generally, starting with some probability distribution $\Pr[x]$, and assuming a random sample $(x_1, x_2, \ldots, x_n)$ of size n from $\Pr[x]$, the theory of orders statistics can be used to calculate the probability that the ith largest value $x_{i:n}$, $i = 1, \ldots n$, in this sample is equal to some value X.

6.4.1 Condensation time scaling

Given the (empirical) probability distribution of the t_c values on the base case lattice of size N, and given k independent sublattices of size N, the probability distribution of the maximum value t_c^{max} of k observed t_c values can be derived using order statistics. In particular, if $F(t_c)$ is the cumulative distribution function (CDF) of the t_c values

on lattice size N, then the probability that the maximum value t_c^{max} of k observed t_c values is equal to i, is given by

$$\Pr[t_c^{max} = i] = [F(i)]^k - [F(i-1)]^k$$

In words, the probability that $t_c^{max} = i$ is equal to the probability that all k observed t_c values are less than or equal to i minus the probability that all k observed t_c values are less than or equal to $i-1$ (since at least one of the k observed t_c values has to be exactly equal to i). The expected value for the condensation time $t_c(k)$ on a lattice of size $k \times N$ is then given by

$$E[t_c(k)] = \sum_{i=0}^{\infty} i \times \left([F(i)]^k - [F(i-1)]^k\right)$$

In figure 6.12, left plot ($k = 1$), the empirical probability distribution of t_c on a lattice of size $N = 100$ is shown for $\phi_{\text{ev-be}}$. This distribution is easily converted into a CDF $F(t_c)$, which can then be used in the above expressions. The plot on the right in figure 6.12 shows another empirical probability distribution (indicated by the bars), but this time on a lattice of size 500 ($k = 5$). The dots connected by a solid line show the probability distribution derived from the order statistics expression above for $\Pr[t_c^{max} = i]$ using the empirical probability distribution on lattice size $N = 100$. As the figure shows, the theory correctly predicts the observed distribution. A χ^2-test yields a p-value slightly above 0.05. In other words, using a significance level of 0.05, there is not enough evidence to reject the hypothesis that the observed and calculated distributions are the same. Results for other values of k are similar.

Using the second expression above for the expected value $E[t_c(k)]$, and the CDF derived from the distribution shown in figure 6.12 (left), the expected values of t_c for various values of k can be calculated. Figure 6.13 shows the results. The solid line shows observed $\overline{t_c}$ values, averaged over 10,000 random ICs. The dashed line shows the predicted values. Again, there is an excellent agreement.

Concluding, predicting the scaling of t_c with lattice size using the theory of order statistics works very well for $\phi_{\text{ev-be}}$. Given a probability distribution of t_c values

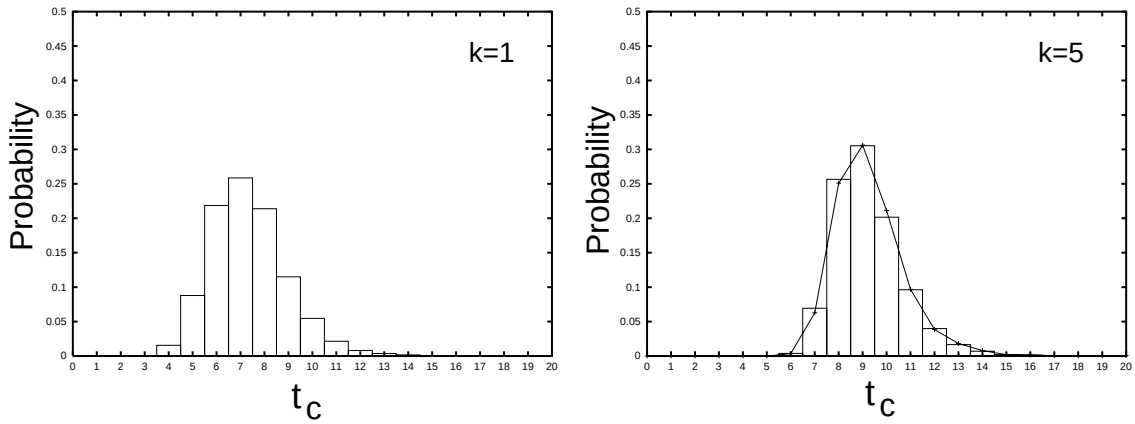

Figure 6.12: Left plot: The empirical probability distribution of t_c on a lattice of size $N = 100$ ($k = 1$). Right plot: The empirical probability distribution of t_c (bars) on a lattice of size $N = 500$ ($k = 5$), and the directly calculated distribution (dots connected by solid line) using order statistics.

for a base case lattice of size N, the probability distribution and expected value for t_c for larger lattices of size kN can be calculated directly.

6.4.2 Scaling of the number of particles at t_c

Next, the total number of particles n at t_c is calculated directly for lattices of size $k \times N$. A first, naive approach would be to measure the average number of particles $\overline{n}(1)$ at t_c for the base case and then multiply this by k to get the expected number of particles $E[n(k)]$ at t_c for the larger lattice size. However, this overestimates the number of particles for the following reason. In the example illustrated in figure 6.11, sublattice 1 condenses fastest, resulting in a sublattice configuration with two particles. However, these particles annihilate each other before the entire lattice condenses (coinciding with the t_c of sublattice 3). Thus, at the condensation time of the entire lattice, the two particles that originally appeared in sublattice 1 no longer exist.

To calculate directly the expected number of particles at t_c on a lattice of size $k \times N$, a "discount" factor needs to be derived that takes into account the average

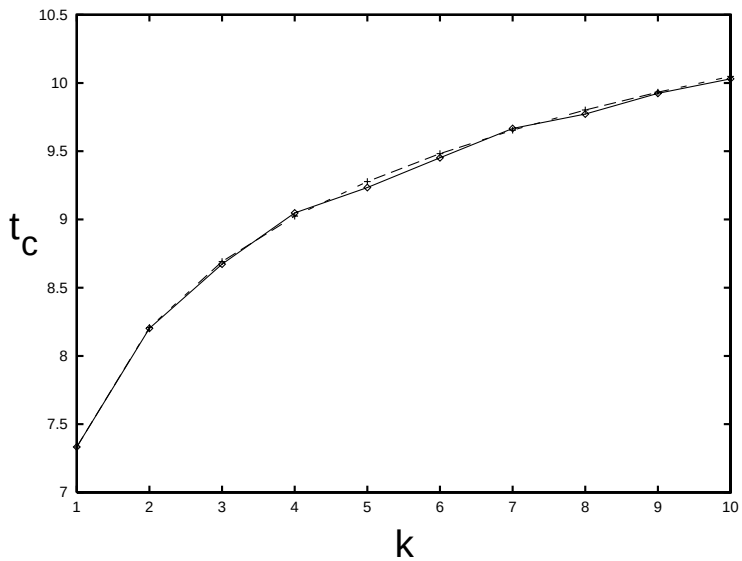

Figure 6.13: Observed values of $\overline{t_c}$ (solid line) and directly calculated values $E[t_c(k)]$ (dashed line) using order statistics.

number of particles that will have annihilated each other between $\overline{t_c}(1)$ and $\overline{t_c}(k)$. First, assume that for the base case $k = 1$ a pair of α and β particles are, on average, $\overline{d}$ sites apart at t_c. Since the velocity of the α particle is $v_\alpha = 1$ and that of the β particle is $v_\beta = -1$, the particles move closer to each other with a relative velocity of $v = 2$. Thus, it takes $\overline{d}/2$ time steps on average for a pair of particles to annihilate.

Given $\overline{n}(1)$ particles at t_c on average, i.e., $\overline{n}(1)/2$ particle pairs, there is one annihilation every $\frac{\overline{d}/2}{\overline{n}(1)/2} = \overline{d}/\overline{n}(1)$ time steps on average. Consequently, there are on average $\overline{n}(1)/\overline{d}$ annihilations per time step. This give gives $\overline{n}(1)t/\overline{d}$ annihilations in t time steps on average. So, during the time $t = \overline{t_c}(k) - \overline{t_c}(1)$, a fraction $2(\overline{t_c}(k) - \overline{t_c}(1))/\overline{d}$ of the original $\overline{n}(1)$ particles at $\overline{t_c}(1)$ have been annihilated, since one annihilation eliminates two particles. Thus, the expected number of particles $E[n(k)]$ at the condensation time on a lattice of size $k \times N$ is given by

$$E[n(k)] = \overline{n}(1) \times k \times \left[1 - \frac{2(\overline{t_c}(k) - \overline{t_c}(1))}{\overline{d}}\right]$$

This is the original "naive" calculation, but including the discount for the fraction of particles that have annihilated each other between $\overline{t_c}(1)$ and $\overline{t_c}(k)$.

To apply this expression, first $\overline{n}(1)$, $\overline{t_c}(1)$, and $\overline{d}$ need to be measured empirically for the base case. Measured over a set of 10,000 random ICs on a lattice of size $N = 100$, these values for $\phi_{\text{ev-be}}$ are: $\overline{n}(1) = 1.568$, $\overline{t_c}(1) = 7.333$, and $\overline{d} = 50$. For $\overline{t_c}(k)$, the previously calculated values using order statistics are used (i.e., for $\overline{t_c}(k)$ the value $E[t_c(k)]$ is used).

Figure 6.14 shows the result of using the above expression to predict the average number of particles at $\overline{t_c}$ for larger lattices. The solid line indicates empirically measured averages $\overline{n}(k)$, and the dashed line indicates the predicted values $E[n(k)]$. In fact, the prediction is so accurate that the dashed line falls right on top of the solid line, and the two lines cannot be distinguished anymore.

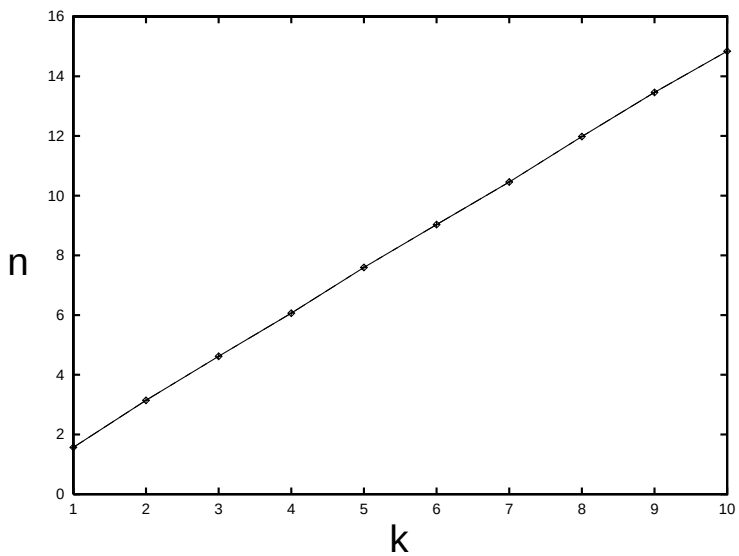

Figure 6.14: Observed values for $\overline{n}(k)$ (solid line) and predicted values $E[n(k)]$ (dashed line). The two lines overlap each other exactly.

Recall that in the computational complexity analysis in section 4.5.2, it was assumed that $n \ll N$. Fitting a linear relation to the data in figure 6.14 results in the following functional form:

$$n = 0.0933 + 0.0147N$$

where $N = 100k$. So, $\alpha \approx 0.015$, and thus n is indeed much smaller than N, about

two orders of magnitude. Empirical measurements for the number of particles n at t_c as a function of the lattice size N for other evolved CAs give similar results. For example, for ϕ_{dens5}, $\alpha = 0.016$ with a correlation coefficient of 0.99. Similarly, for ϕ_{sync5}, $\alpha = 0.028$, also with a correlation coefficient of 0.99.

Concluding, this section has shown that given the probability distributions of the condensation time t_c and the total number of particles n at t_c for some base lattice size, the expected values for t_c and n on larger lattice sizes can be calculated directly using the theory of order statistics. So, the approximation of the PPD at t_c, which is the only part in a CA's particle model that was lattice size dependent, can be scaled accurately according to the lattice size N. As a consequence, the CA's particle model is now completely independent of the lattice size.

However, the ability to predict t_c and n on larger lattices is shown for a CA with a relatively simple particle logic. For this CA, the assumption that the sublattices in a larger lattice are independent of each other holds up quite well (and the theory of order statistics is based on this assumption). In more complicated CAs, this assumption might not hold up as well. Consequently, the expressions for calculating $E[t_c]$ and $E[n(k)]$ will become more complicated.

6.5 Conclusions

In the previous chapter, the results generated by the class of particle models provided a better understanding of the relation between dynamics and emergent computation in evolved CAs. The results in this chapter, in contrast, provide a better understanding of the class of models itself. First, it was proved, in the most basic setting, that a CA's particle model correctly models the CA's dynamics. This provides an important fundamental proof of the correctness and completeness of the particle models.

Next, it was shown that in some specific cases an accurate *and* concise approximation of the PPD at t_c can be found. Using empirically measured probability

distributions of occurrences of domains and particles at t_c, the PPD at t_c of two evolved CAs was modeled accurately. These PPD models can then be used in the respective particle models of these CAs to predict their performances. Furthermore, using these concise PPD models, an expression was derived for calculating the performance predictions of these CAs. This way, the performances of these CAs can be predicted directly, without having to run their respective particle models.

Finally, scaling issues were addressed. In particular, expressions for calculating the expected values for t_c and the number of particles at t_c as a function of the lattice size N were derived using the theory of order statistics, for one particular evolved CA. With these expressions, the average t_c and average number of particles at t_c can be predicted very accurately for larger lattice sizes. This shows that an approximation of the PPD at t_c in this CA's particle model needs to be constructed only once, based on one particular lattice size, and can then be scaled appropriately for use on other lattice sizes. This makes the particle model completely independent of the lattice size.

Chapter 7

Conclusions and Discussion

In this final chapter, a summary of the research presented in this dissertation and the main conclusions following from it are provided. This is followed by some discussion of issues directly related to this work and its conclusions. Finally, some suggestions for future work are presented.

7.1 Summary and conclusions

The main goal of this dissertation has been to study the relation among dynamics, emergent computation, and evolution in cellular automata. This study was motivated by the more general questions of (1) how dynamics (in particular spatio-temporal patterns) and emergent computation are related in decentralized spatially extended systems, and (2) how evolution produces emergent computation in decentralized spatially extended systems.

The evolving cellular automata (EvCA) framework forms an ideal model for studying these relations. In the EvCA framework, cellular automata (CAs) are evolved by a genetic algorithm (GA) to perform computational tasks requiring global information processing. The CAs evolved for these tasks show emergent patterns in their dynamics, which can be classified and described in terms of domains, particles, and particle interactions, using the computational mechanics (CM) framework. These domains, particles, and particle interactions provide a means for the evolved CAs to perform the emergent computation that is necessary for performing the given task. In this sense, they can be interpreted as the computational strategy of the CA.

Chapter 3 provided an overview of previous results from EvCA experiments on evolving CAs for the density classification and the global synchronization–1 tasks. Furthermore, it presented new results on evolving CAs on two additional tasks, both related to the original global synchronization task. In the CAs evolved for these additional tasks, patterns similar to those of previously evolved CAs can be observed. These patterns can also be classified and described in terms of domains,

particles, and particle interactions, and interpreted as the CAs computational strategy. In fact, the computational strategy of the best evolved CA for the global synchronization–2 task, $\phi_{\text{sync-2b}}$, is equivalent to that of ϕ_{sync5}, the best evolved CA for the global synchronization–1 task. However, as the best evolved CA on the global synchronization–3 task, $\phi_{\text{sync-3b}}$, shows, there are other computational strategies with which global synchronization tasks can be performed equally well.

In chapter 4, a new class of particle models for analyzing the computational strategies in evolved CAs in terms of domains, particles, and particle interactions was introduced. This class of particle models provides a high-level description of a CA's dynamics and captures the notion of computational strategy in evolved CAs. It consists of (1) an approximation of the particle probability distribution (PPD) at the condensation time t_c, i.e., a probability distribution of where and how often particles occur after an initial transient time, and (2) the CA's particle catalog, a summary of the domains, particles, and particle interactions occuring in a CA's dynamics. Such a particle model provides an algorithmic procedure for simulating the CA's computational strategy. This procedure can then be used to predict the CA's computational performance on a given task.

This new class of models was shown to have a computational complexity that is lower than that of a CA. The memory requirements for running a CA's particle model are about the same, or in some cases even less, than the memory requirements for running the CA itself. The time complexity of running a CA is quadratic in the lattice size N. The time complexity of a CA's particle model is generally also quadratic, but in the number n of particles in the lattice. The analysis in chapter 6 showed that this number n is about 2 orders of magnitude smaller than N. Furthermore, in many cases the time complexity of a particle model becomes linear depending on the kind of particle interactions (in particular, how many new particles result from an interaction) and on how often interactions happen on average. Finally, running a CA is generally not efficiently parallelizable, whereas running a CA's particle model is, at

least in a number of cases (again depending on the kind of particle interactions). It can thus be concluded that a CA's particle model forms a more concise and efficient description of a CA's dynamics compared to explicitly constructing the space-time configurations of the CA at each time step.

In chapter 5 it was shown that the class of particle models is capable of accurately predicting the computational performance of the evolved CAs. This provides a direct relation between the dynamics (computational strategy) of a CA and the emergent computation it performs. This relation answers, at least in the context of CAs, the first main question above: What is the relation between dynamics and emergent computation in decentralized spatially extended systems? The CA generates emergent patterns, which can be described in terms of domains, particles, and particle interactions, which it then uses to perform the emergent computation necessary to perform a given task. In particular, the particles can be interpreted as carriers of information, and the particle interactions are the loci of information processing.

Furthermore, in chapter 5 it was also shown that the particle models can be used to analyze the differences in the computational strategies between CAs, and how these differences contribute to differences in their performances. In particular, the properties of particles (such as velocity and how they interact with other particles) and their contribution to a CA's performance can be investigated in isolation, which generally is not possible in the CA itself. This investigation has led to a direct understanding of how and why the computational strategy of one CA is better than that of another, evolutionarily related, CA. This understanding answers, again at least in the context of CAs, the second main question: How does evolution produce emergent computation in decentralized spatially extended systems? In the evolution of CAs, the GA manipulates the bits in the lookup tables of the CAs, which causes changes in the emergent patterns observed in the CAs. A change that causes an increase in a CA's computational performance is preserved, and deleterious changes are selected against.

Chapter 6 then provided a further investigation of some of the properties of the class of particle models. First, it was proved that in the case where a CA does not violate any of the simplifying assumptions in the models, the CA's particle model forms a complete description of the CA's dynamics. In other words, there is a direct mapping between the CA's emergent dynamics and the model description of that dynamics, and there is no behavior in the CA that is not accounted for in the CA's model.

Next, it was argued that finding a general and concise approximation of the PPD at t_c is nontrivial. An example of a general method was presented, and it was then shown that the results form this method were far from accurate. However, some examples were given of concise but very specific approximations. These approximations work only for a limited class of CAs since they incorporate some assumptions about a CA's behavior which do not hold in general. It was then shown that, given these consice but specific approximations, it is possible to derive an expression for predicting a CA's performance directly, without having to run the particle model simulation. These direct predictions are in close agreement with those generated by a CA's particle model.

Finally, also in chapter 6, scaling issues with respect to a CA's lattice size were addressed. Expressions were derived for predicting the distribution of t_c values and for the (average) number of particles at t_c for larger lattice sizes, based on their distributions on a base case lattice size. The predictions resulting from these expressions are extremely accurate when compared to the corresponding values as measured in the CA. This makes a CA's particle model independent of the lattice size N, since in principle the PPD at t_c can be scaled with N.

In summary, the results obtained with the class of particle models show that it is possible to give an accurate high-level description of an evolved CA's dynamics in terms of domains, particles, and particle interactions. This high-level description can then be interpreted as a computational strategy, where particles carry information

across the lattice, and where particle interactions give rise to the processing and exchanging of information. Finally, this high-level description can be used to predict the computational performances of evolved CAs, and to relate changes in computational strategies that occur during the evolution to changes in computational performances. This way, these results provide a better and direct understanding of the relation among dynamics, emergent computation, and evolution in cellular automata.

7.2 Discussion

One issue that has not been addressed so far, and one that forms a research project on its own, is the relation between a CA's computational strategy, and changes therein, and the bits in the CA's lookup table. In other words, how do the specific bit values in a CA's lookup table give rise to domains, particles, and particle interactions? Furthermore, how do changes in these bit values cause changes in the CA's emergent dynamics?

It is generally very difficult to predict the emergent behavior of a CA from its lookup table (LUT). Of course, this is exactly why this behavior is called *emergent*, i.e., although it is necessarily an indirect consequence of the LUT, it cannot be directly derived from the local rules of the constituent components in the system. Nonetheless, one can go the other way. Given the domains and particles in a CA's emergent dynamics, it can be determined what bits in its LUT are "necessary" for supporting these structures. An example is given in figure 7.1. This figure shows a space-time diagram of ϕ_{parent} (the same one as was shown in figure 5.12), with its three particles labeled. Below the space-time diagram, the bit string representing ϕ_{parent}'s LUT is shown. Below that, the bits that are necessary for supporting each of the three particles are given. In this representation, a dot indicates that the bit is not necessary for supporting the particle (i.e., it can be either 0 or 1).

As the figure shows, the three particles require different sets of bits in the LUT.

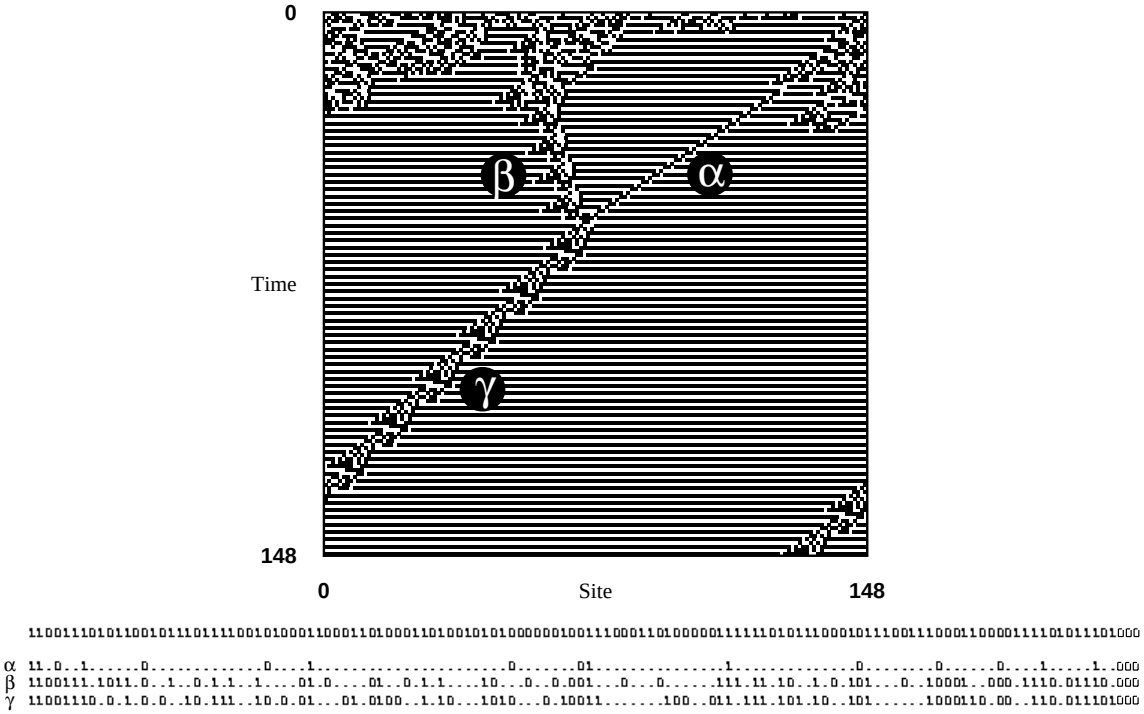

Figure 7.1: A space-time diagram of ϕ_{parent}, with its three particles labeled. The bit string representing ϕ_{parent}'s LUT is given below the space-time diagram. Below that, the bits necessary for supporting each of the three particles are given.

Some bits are uniquely set for one particle, other bits are shared between particles, and some bits are not required for any particles. To determine the importance of each bit in a CA's computational strategy, the LUT bits can be mutated, one by one, to see what changes are induced in the CA's dynamics and whether the CA's performance changes significantly. An analysis based on such single-bit mutations was done in [Hor97] for ϕ_{parent} and ϕ_{child}.

A more detailed analysis of the importance of individual bits in the CA's LUT is presented in [SHC], where a number of CAs that were evolved for the density classification task are analyzed. It is shown there, for example, that there exist complicated, nonlinear interactions between groups of LUT bits, so called *epistatic* interactions. For example, two bits that, when mutated individually, do not cause a change in a CA's performance can cause a significant change when mutated together. Or vice versa, two bits that cause a significant change individually, might not cause

a change when mutated together. The goal of this type of epistatic analysis is a better understanding of the relation between the bit values in a CA's LUT and its (emergent) dynamics.

Another issue worth at least a brief discussion is the generality of the occurrence of domains and particles in CAs that are evolved to perform a certain computational task. The results presented in chapter 3 on the additional tasks show that in CAs evolved for these new tasks domains and particles occur as well, just as in the CAs evolved for the original two tasks. However, in all the tasks studied here the answer states that the CA must converge to are regular repeating configurations themselves. It is therefore perhaps not surprising that the evolved CAs use regular structures such as domains and particles to reach such an answer state. One could ask whether this will still be the case when CAs are evolved on a computational task that does not have any regular configurations as answer states. One example might be evolving CAs to generate (pseudo) random numbers, where the lattice configurations after a certain time step M are interpreted as binary numbers. These binary numbers generated by the CA then have to pass certain tests for randomness. The more random this sequence of binary numbers is, the higher the performance of the CA will be.

Generally, however, when performing a computation a certain amount of information needs to be stored and transferred. As the presented analyses have shown, the domains and particles observed in the evolved CAs can be interpreted as performing this information storage and transfer. Thus, it seems plausible that structures like domains and particles will often be used when a CA is required to perform global information processing. Furthermore, domains and particles appear in many CAs, regardless of whether they are required to perform any kind of computation. These structures are inherent in systems like CAs and can be manipulated by evolution to be put to a certain use, as the results presented in this dissertation show.

The particle models as introduced in chapter 4 incorporate a number of simplifying assumptions. As the results in chapter 5 showed, these assumptions can lead to

mistakes in simulating a CA's dynamics, which in turn leads to minor discrepancies between the model and CA performances. One assumption that is implicit in the particle models and which has not been mentioned explicitly yet, is that the particles have a constant velocity. This assumption holds for all CAs that have been analyzed here and thus has not caused any problems. However, it happens occasionaly that a CA is evolved that has a so called *chaotic* domain for which the particles that form its boundaries do not have a constant velocity, but effectively move randomly.

A well-known example of a CA with a chaotic domain is ECA 18. A space-time diagram of this CA was shown in figure 2.2. As it turns out, the particles (called dislocations for this CA) behave in a way similar to annihilating diffusive particles. An example of an evolved CA with a chaotic domain is given in in figure 7.2. The CA in this figure is the best CA in the final population of one particular GA run on the global synchronization–2 task. Its fitness in that generation was 0.83.

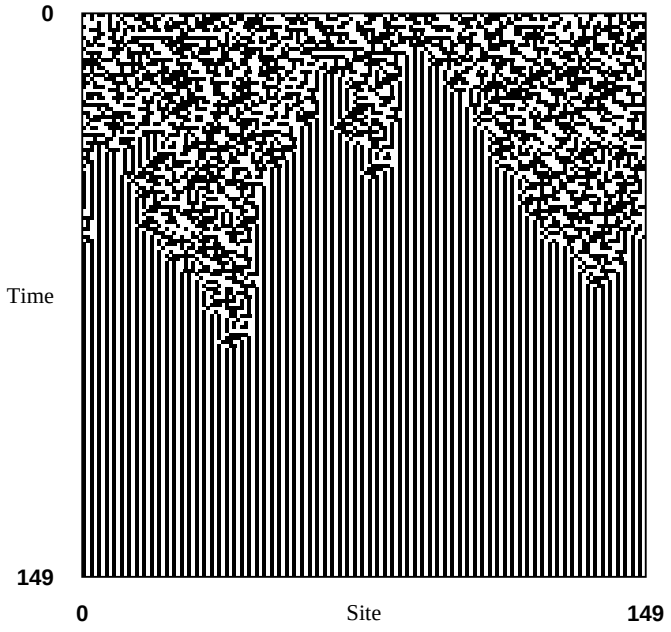

Figure 7.2: A space-time diagram of a CA evolved for the global synchronization–2 task. It has a fitness of 0.83.

With the current implementation of the particle models, the dynamics of this CA cannot be simulated accurately, since the boundaries between the different do-

mains do not have a constant velocity. However, perhaps an average velocity can be used. Or maybe one can estimate probabilities for moving left or right at each time step, and adjust the particle model to calculate an expected distance over which a domain boundary will have moved after a certain number of time steps. However, the occurrences of CAs like the one shown in figure 7.2 in evolving CAs to perform computational tasks are rare. And as just suggested, the current implementation of the particle models needs only a minor modification to be able to simulate the dynamics of such CAs as well.

In general, however, the particle model of an evolved CA forms an accurate description of its dynamics and provides a direct relation between the dynamics and the emergent computation in evolved CAs. Given this relation for CAs, one could ask: How much have we learned about the relation between dynamics and emergent computation in decentralized spatially extended systems in general? Do other spatially extended systems also use particle-like structures to store and transfer information and particle interactions to process and exchange this information? For now, these questions remain largely unanswered. However, some indications for a possible answer can be provided.

The CAs studied in this dissertation are all one-dimensional CAs. Most spatially extended systems in nature, and the patterns that emerge in them, are at least two-dimensional. Therefore, it is not directly obvious how to translate the results for these one-dimensional CAs to other two-or-higher-dimensional systems. However, there have been several studies of spatialy extended systems and models thereof, where also one-dimensional versions of these models or one-dimensional cross-sections of such systems or models have been studied. The results of these studies can thus be compared to the CA results presented here, at least qualitatively. Next, some examples of such studies are listed.

As a first example, in [BJP+95] a well-known model of chemical reaction systems, the Brusselator (BX) model, is studied. This model generates patterns that are

observed in experimental chemical reaction systems. In [BJP+95], several space-time diagrams resulting from a one-dimensional BX model are presented. These diagrams clearly show the occurrence of structures very similar to domains and particles in CAs. The domains in this system, stationary space periodic concentration patterns, are called Turing structures. The particles, forming the boundaries between the domains, are called "fronts". These fronts collide and interact with each other, forming other fronts. It has been argued that similar processes (i.e., the formation of Turing structures) underly certain kinds of biological pattern formation.

Another example, in a similar realm, can be found in [TH88]. In this paper, the Greenberg-Hastings model, a cellular automaton model of a reaction-diffusion system, is studied. In two dimensions, this model often gives rise to spiral waves similar to those observed in experimental reaction-diffusion systems. The authors also study a one-dimensional version of this model and present space-time diagrams showing the dynamics of the model. In these space-time diagrams, again domain- and particle-like structures can be observed.

As a next example, in [RTWE98] a spatially structured network of inhibitory neurons is studied. The space-time diagrams generated by a one-dimensional version of this network show "propagating activity patterns", as the authors call them. In fact, these activity patterns seem to behave as traveling particles between stable domains of less active neurons.

An example which is more in the context of an evolving system is the cellular automaton model of self-replicative molecules that give each other cyclic-catalytic support. This model was studied in [BH91]. The two-dimensional models give rise to (emergent) spiral waves. The authors show that because of these spiral waves, selection no longer exclusively takes place at the level of the individual molecules, but also at the level of the spirals. This model was mainly studied in a two-dimensional context, but the authors also provide space-time diagrams of one-dimensional cross-sections from their model. Again, domain- and particle-like structures can be seen.

As a final example, in [Mei98] an appendix on pattern formation in the development of organisms is provided. In this appendix traveling waves that are observed in aggregating *Dictyostelium* amoebae are discussed. These waves, emerging out of local cell-to-cell interactions, are used by the individual amoebae to decide when and where to aggregate and form a multicellular organism that can then reproduce. A picture of these waves occurring in aggregating amoebae is provided, together with a space-time diagram showing the movements of individual amoebae in a one-dimensional cross-section. Structures reminiscent of domains and particles can be seen here.

These examples show that one-dimensional models or cross-sections of a wide variety of spatially extended systems show structures very similar to the domains and particles observed in the one-dimensional evolved CAs. One possible implication of this observation is that global information processing in such systems is done in a similar way as in the CAs studied here. In some of the examples above, such as those for chemical reaction and reaction-diffusion systems, it is not directly clear whether, or what kind of, information processing is going on in the system. However, in systems like aggregating amoebae and neural networks there is clearly some kind of decision making or information processing going on. It would be very interesting to investigate whether these processes are indeed similar to those in the evolved CAs.

To end this discussion, two other areas where the results of the study presented here can be relevant are mentioned. First, cellular automata are a simple model of parallel computation (see, e.g., [Hil84, TM87]). Designing algorithms for parallel computers is generally difficult for at least two reasons: (1) achieving an equal distribution of the workload over the different processors and (2) minimizing communication overhead between. The EvCA framework shows, in a simple setting, that it is possible to overcome some of these difficulties by evolving computational strategies (algorithms) for CAs (parallel computers). The results presented in this dissertation have provided a direct understanding of how these algorithms, evolved on parallel computers, perform the given computational task. Therefore, this under-

standing could be relevant to the design of collective computation in multiprocessor systems in general. Alternatively, it should be possible to evolve algorithms for parallel computers on more realistic problems as well.

A second area of relevance is that of predicting a CA's behavior from the initial configuration (IC). As discussed in chapter 4, predicting the state s_t^i of a cell i at a certain time t is a P-complete problem. For some special CAs, in particular additive and quasi-linear CAs, predicting s_t^i can be done quickly, in $\mathcal{O}(\log t)$ or $\mathcal{O}(\log^2 t)$ time, placing them in the complexity class NC of efficiently parallelizable problems. However, with a CA's particle model, it might also be possible to quickly predict s_t^i. It was shown in chapter 4 that in some cases a CA's particle model can be efficiently parallelized. Thus, when a particle model accurately simulates a CA's dynamics and is efficiently parallelizable, it can be used to quickly predict s_t^i without having to run the entire CA.

7.3 Future work

There are many directions in which to go, and the suggestions made here are therefore necessarily incomplete. However, they indicate what still needs to be done for a complete understanding of the relation among dynamics, emergent computation, and evolution in decentralized spatially extended systems.

First of all, CAs should be evolved for a wider range of different computational tasks. As mentioned in the previous section, it is unknown whether domains and particles will still occur when CAs are evolved for a task that does not have regular, repeating configurations as answer states. This dissertation presented results on evolving CAs on two new tasks, but these tasks were directly related to the original global synchronization task. It would be interesting to come up with new and different tasks.

How exactly domains and particles are formed during the condensation phase is

still not well understood. How much of a CA's computational performance is a result of what is going on during this phase, and how much is primarily due to the particle logic after t_c? A better understanding of the dynamics during the condensation phase is necessary for a complete understanding of an evolved CA's dynamics. Related to this is the problem of finding a general and concise approximation of the PPD at t_c. A better understanding of the dynamics during the condensation phase might lead to such a general approximation.

Of course, other questions that are still left open in this dissertation are worth investigating. These include expressions for directly predicting a CA's computational performance that works for general CAs, and extending the scaling results to the general case. For these problems, some CA-specific solutions were presented in this dissertation. Solving these problems for the general case remains to be done.

Another interesting extension would be to evolve complete particle logics, instead of evolving CA update rules. Of course, the question then is whether these evolved particle logics can be re-implemented in an actual CA. As an example, in [HC97] a CA is constructed that implements one particular particle logic. Also, when the number of states (i.e., the alphabet size) of the CA is not restricted to any upper limit, then every (finite) particle logic can be implemented directly in some CA. However, when one is restricted to, say, 2-state CAs, this generally becomes a nontrivial problem.

Finally, as alluded to in the previous section, it would be an interesting next step to see if the computational analyses in terms of domains, particles, and particle interactions can be applied to decentralized spatially extended systems, or models thereof, other than CAs. Can we build models similar to the particle models introduced here to study emergent computation in other systems? If so, then hopefully the work presented here will serve as a first step towards a more complete understanding of the relation among dynamics, emergent computation, and evolution in decentralized spatially extended systems in general.

Appendix A

Particle Catalogs

In this appendix, the particle catalogs of the CAs that are analyzed in this dissertation are presented, as far as they have not been presented in the main text yet. These catalogs are in the same format as the particle catalog of ϕ_{sync5} as presented in section 3.5.

A.1 Density classification

Below are the particle catalogs of the five CAs that occurred during a GA run on the density classification task: ϕ_{dens1} to ϕ_{dens5}.

ϕ_{dens1}				
Domains Λ				
Label	Regular language			
Λ^w	0^*			
Λ^b	1^*			
Particles P				
Label	Boundary	p	d	v
α	$\Lambda^w\Lambda^w$	1	2	2
β	$\Lambda^w\Lambda^w$	3	4	4/3
γ	$\Lambda^w\Lambda^b$	1	1	1
δ	$\Lambda^w\Lambda^b$	1	-1	-1
ε	$\Lambda^b\Lambda^w$	3	4	4/3
η	$\Lambda^b\Lambda^w$	1	2	2
Interactions I				
$\alpha+\beta\rightarrow\gamma+\varepsilon$		$\alpha+\gamma\rightarrow\gamma$		
$\alpha+\delta\rightarrow\gamma$		$\beta+\gamma\rightarrow\gamma$		
$\beta+\delta\rightarrow\gamma$		$\eta+\beta\rightarrow\varepsilon$		
$\varepsilon+\gamma\rightarrow\emptyset$		$\eta+\gamma\rightarrow\emptyset$		
$\varepsilon+\delta\rightarrow\emptyset$		$\eta+\delta\rightarrow\emptyset$		

ϕ_{dens2}				
Domains Λ				
Label	Regular language			
Λ^w	0^*			
Λ^b	1^*			
$\Lambda^\#$	$\{(01)^*,(10)^*\}$			
Particles P				
Label	Boundary	p	d	v
α	$\Lambda^w\Lambda^b$	1	1	1
β	$\Lambda^b\Lambda^w$	2	1	1/2
γ	$\Lambda^w\Lambda^\#$	1	-1	-1
δ	$\Lambda^\#\Lambda^w$	1	-3	-3
ε	$\Lambda^b\Lambda^\#$	1	3	3
η	$\Lambda^\#\Lambda^b$	1	1	1
Interactions I				
$\alpha+\beta\rightarrow\emptyset$		$\beta+\gamma\rightarrow\varepsilon$		
$\eta+\beta\rightarrow\delta$		$\gamma+\delta\rightarrow\emptyset$		
$\varepsilon+\delta\rightarrow\beta$		$\varepsilon+\eta\rightarrow\emptyset$		

ϕ_{dens3}				
Domains Λ				
Label	Regular language			
Λ^w	0^*			
Λ^b	1^*			
$\Lambda^\#$	$\{(01)^*, (10)^*\}$			
Particles P				
Label	Boundary	p	d	v
α	$\Lambda^w\Lambda^b$	1	1	1
β	$\Lambda^b\Lambda^w$	3	1	1/3
γ	$\Lambda^w\Lambda^\#$	1	-1	-1
δ	$\Lambda^\#\Lambda^w$	1	-3	-3
ε	$\Lambda^b\Lambda^\#$	2	6	3
η	$\Lambda^\#\Lambda^b$	1	1	1
Interactions I				
$\alpha + \beta \to \emptyset$		$\beta + \gamma \to \varepsilon$		
$\eta + \beta \to \delta$		$\gamma + \delta \to \emptyset$		
$\varepsilon + \delta \to \beta$		$\varepsilon + \eta \to \emptyset$		

ϕ_{dens4}				
Domains Λ				
Label	Regular language			
Λ^w	0^*			
Λ^b	1^*			
$\Lambda^\#$	$\{(01)^*, (10)^*\}$			
Particles P				
Label	Boundary	p	d	v
α	$\Lambda^w\Lambda^b$	1	1	1
β	$\Lambda^b\Lambda^w$	1	0	0
γ	$\Lambda^w\Lambda^\#$	1	-1	-1
δ	$\Lambda^\#\Lambda^w$	1	-3	-3
ε	$\Lambda^b\Lambda^\#$	1	3	3
η	$\Lambda^\#\Lambda^b$	1	1	1
Interactions I				
$\alpha + \beta \to \emptyset$		$\beta + \gamma \to \varepsilon$		
$\eta + \beta \to \delta$		$\gamma + \delta \to \emptyset$		
$\varepsilon + \delta \to \beta$		$\varepsilon + \eta \to \emptyset$		

ϕ_{dens5}				
Domains Λ				
Label	Regular language			
Λ^w	0^*			
Λ^b	1^*			
$\Lambda^\#$	$\{(01)^*, (10)^*\}$			
Particles P				
Label	Boundary	p	d	v
β	$\Lambda^b\Lambda^w$	1	0	0
γ	$\Lambda^w\Lambda^\#$	1	-1	-1
δ	$\Lambda^\#\Lambda^w$	1	-3	-3
ε	$\Lambda^b\Lambda^\#$	1	3	3
η	$\Lambda^\#\Lambda^b$	1	1	1
Interactions I				
$\beta + \gamma \to \varepsilon$		$\eta + \beta \to \delta$		
$\gamma + \delta \to \emptyset$		$\varepsilon + \delta \to \beta$		
$\varepsilon + \eta \to \emptyset$				

A.2 Global synchronization–1

Next, the particle catalogs of the CAs that were evolved on the global synchronization–1 task are presented. First, the catalogs for ϕ_{sync1} to ϕ_{sync4} are given. The particle catalog of ϕ_{sync5} was already given in section 3.5. Then, the catalogs of ϕ_{parent} and ϕ_{child} are given.

ϕ_{sync1}				
Domains Λ				
Label	Regular language			
Λ^s	$\Lambda^s_0 = 0^*$, $\Lambda^s_1 = 1^*$			
Particles P				
Label	Boundary	p	d	v
α	$\Lambda^s_0\Lambda^s_1, \Lambda^s_1\Lambda^s_0$	2	-1	-1/2
β	$\Lambda^s_0\Lambda^s_0, \Lambda^s_1\Lambda^s_1$	4	-1	-1/4
γ	$\Lambda^s_0\Lambda^s_0, \Lambda^s_1\Lambda^s_1$	8	-1	-1/8
δ	$\Lambda^s_0\Lambda^s_0, \Lambda^s_1\Lambda^s_1$	2	0	0
Interactions I				
$\beta + \alpha \rightarrow \alpha$		$\gamma + \alpha \rightarrow \alpha$		
$\delta + \alpha \rightarrow \alpha$		$\gamma + \beta \rightarrow \beta$		
$\delta + \beta \rightarrow \beta$		$\delta + \gamma \rightarrow \beta$		

ϕ_{sync2}				
Domains Λ				
Label	Regular language			
Λ^s	$\Lambda^s_0 = 0^*$, $\Lambda^s_1 = 1^*$			
Particles P				
Label	Boundary	p	d	v
α	$\Lambda^s_0\Lambda^s_1, \Lambda^s_1\Lambda^s_0$	2	-1	-1/2
β	$\Lambda^s_0\Lambda^s_1, \Lambda^s_0\Lambda^s_1$	6	0	0
Interactions I				
$\beta + \alpha \rightarrow \emptyset$				

ϕ_{sync3}				
Domains Λ				
Label	Regular language			
Λ^s	$\Lambda^s_0 = 0^*$, $\Lambda^s_1 = 1^*$			
Particles P				
Label	Boundary	p	d	v
α	$\Lambda^s_0\Lambda^s_1, \Lambda^s_1\Lambda^s_0$	4	-3	-3/4
β	$\Lambda^s_0\Lambda^s_1, \Lambda^s_0\Lambda^s_1$	6	0	0
γ	$\Lambda^s_0\Lambda^s_1, \Lambda^s_0\Lambda^s_1$	12	3	1/4
δ	$\Lambda^s_0\Lambda^s_1, \Lambda^s_0\Lambda^s_1$	2	1	1/2
Interactions I				
$\beta + \alpha \to \emptyset$		$\gamma + \alpha \to \emptyset$		
$\delta + \alpha \to \emptyset$		$\gamma + \beta \to \emptyset$		
$\delta + \beta \to \emptyset$		$\delta + \gamma \to \emptyset$		

ϕ_{sync4}				
Domains Λ				
Label	Regular language			
Λ^s	$\Lambda^s_0 = 0^*$, $\Lambda^s_1 = 1^*$			
Λ^z	$\Lambda^z_0 = (0001)^*$, $\Lambda^z_1 = (1110)^*$			
Particles P				
Label	Boundary	p	d	v
α	$\Lambda^s_1\Lambda^s_0$	-	-	-
β	$\Lambda^z_0\Lambda^s_0, \Lambda^z_1\Lambda^s_1$	2	2	1
γ	$\Lambda^s_0\Lambda^z_1, \Lambda^s_1\Lambda^z_0$	2	-2	-1
δ	$\Lambda^z_0\Lambda^s_1, \Lambda^z_1\Lambda^s_0$	4	-12	-3
μ	$\Lambda^s_0\Lambda^z_0, \Lambda^s_1\Lambda^z_1$	2	6	3
ν	$\Lambda^z_0\Lambda^z_1, \Lambda^z_1\Lambda^z_0$	2	-2	-1
Interactions I				
$\alpha \to \gamma + \beta$		$\beta + \gamma \overset{0.84}{\to} \delta + \mu$		
$\beta + \gamma \overset{0.16}{\to} \nu$		$\mu + \beta \to \emptyset$		
$\gamma + \delta \to \emptyset$		$\mu + \delta \to \delta + \gamma$		
$\nu + \delta \to \beta$		$\mu + \nu \to \gamma$		

ϕ_{parent}				
Domains Λ				
Label	Regular language			
Λ^s	$\Lambda^s_0 = 0^*$, $\Lambda^s_1 = 1^*$			
Particles P				
Label	Boundary	p	d	v
α	$\Lambda^s_0\Lambda^s_1, \Lambda^s_1\Lambda^s_0$	2	-3	-3/2
β	$\Lambda^s_0\Lambda^s_1, \Lambda^s_1\Lambda^s_0$	8	2	1/4
γ	$\Lambda^s_0\Lambda^s_0, \Lambda^s_1\Lambda^s_1$	12	-12	-1
Interactions I				
$\beta + \alpha \overset{0.30}{\to} \emptyset$		$\beta + \alpha \overset{0.30}{\to} \gamma$		
$\beta + \alpha \overset{0.10}{\to} 2\alpha$		$\beta + \alpha \overset{0.30}{\to} \alpha + \beta$		
$\gamma + \alpha \overset{0.84}{\to} \beta$		$\gamma + \alpha \overset{0.16}{\to} \gamma + \beta$		
$\beta + \gamma \overset{0.40}{\to} \alpha$		$\beta + \gamma \overset{0.55}{\to} \beta$		
$\beta + \gamma \overset{0.05}{\to} \alpha + \gamma$				

ϕ_{child}				
Domains Λ				
Label	Regular language			
Λ^s	$\Lambda^s_0 = 0^*$, $\Lambda^s_1 = 1^*$			
Particles P				
Label	Boundary	p	d	v
α	$\Lambda^s_0\Lambda^s_1, \Lambda^s_1\Lambda^s_0$	2	-3	-3/2
β	$\Lambda^s_0\Lambda^s_1, \Lambda^s_0\Lambda^s_1$	8	0	0
Interactions I				
$\beta + \alpha \overset{0.74}{\to} \emptyset$		$\beta + \alpha \overset{0.13}{\to} 2\alpha$		
$\beta + \alpha \overset{0.13}{\to} \alpha + \beta$				

A.3 Global synchronization–2

The particle catalogs of the two CAs that were evolved on the global synchronization–2 task, $\phi_{\text{sync-2a}}$ and $\phi_{\text{sync-2b}}$, are presented below.

$\phi_{\text{sync-2a}}$				
Domains Λ				
Label	Regular language			
Λ^s	$\Lambda^s_0 = (01)^*$, $\Lambda^s_1 = (10)^*$			
Particles P				
Label	Boundary	p	d	v
α	$\Lambda^s_0\Lambda^s_1, \Lambda^s_1\Lambda^s_0$	2	0	0
β	$\Lambda^s_0\Lambda^s_1, \Lambda^s_1\Lambda^s_0$	2	2	1
γ	$\Lambda^s_0\Lambda^s_1, \Lambda^s_1\Lambda^s_0$	1	2	2
δ	$\Lambda^s_0\Lambda^s_1, \Lambda^s_1\Lambda^s_0$	2	4	2
ε	$\Lambda^s_0\Lambda^s_1, \Lambda^s_1\Lambda^s_0$	4	2	1/2
η	$\Lambda^s_0\Lambda^s_1, \Lambda^s_1\Lambda^s_0$	1	2	2
Interactions I				
$\beta + \alpha \to \emptyset$		$\gamma + \alpha \to \emptyset$		
$\delta + \alpha \to \emptyset$		$\varepsilon + \alpha \to \emptyset$		
$\eta + \alpha \to \emptyset$		$\gamma + \beta \to \emptyset$		
$\delta + \beta \to \emptyset$		$\beta + \varepsilon \overset{0.50}{\to} \emptyset$		
$\beta + \varepsilon \overset{0.50}{\to} \alpha + \gamma$		$\eta + \beta \to \emptyset$		
$\gamma + \varepsilon \to \emptyset$		$\delta + \varepsilon \overset{0.83}{\to} \emptyset$		
$\delta + \varepsilon \overset{0.17}{\to} \alpha + \gamma$		$\eta + \varepsilon \to \emptyset$		

$\phi_{\text{sync-2b}}$				
Domains Λ				
Label	Regular language			
Λ^s	$\Lambda^s_0 = 0^*$, $\Lambda^s_1 = 1^*$			
Λ^z	$\Lambda^z_0 = (0011)^*$, $\Lambda^z_1 = (1110)^*$			
Particles P				
Label	Boundary	p	d	v
α	$\Lambda^s_0\Lambda^s_1, \Lambda^s_1\Lambda^s_0$	-	-	-
β	$\Lambda^z_0\Lambda^s_0, \Lambda^z_1\Lambda^s_1$	2	2	1
γ	$\Lambda^s_0\Lambda^z_1, \Lambda^s_1\Lambda^z_0$	2	-2	-1
δ	$\Lambda^z_0\Lambda^s_1, \Lambda^z_1\Lambda^s_0$	2	-6	-3
μ	$\Lambda^s_0\Lambda^z_0, \Lambda^s_1\Lambda^z_1$	2	6	3
ν	$\Lambda^z_0\Lambda^z_1, \Lambda^z_1\Lambda^z_0$	2	-2	-1
Interactions I				
$\alpha \to \gamma + \beta$		$\beta + \gamma \overset{0.88}{\to} \delta + \mu$		
$\beta + \gamma \overset{0.12}{\to} \nu$		$\mu + \beta \to \emptyset$		
$\gamma + \delta \to \emptyset$		$\mu + \delta \to \gamma + \beta$		
$\nu + \delta \to \beta$		$\mu + \nu \to \gamma$		

A.4 Global synchronization–3

Finally, the particle catalogs of the two CAs that were evolved on the global synchronization–3 task, $\phi_{\text{sync-3a}}$ and $\phi_{\text{sync-3b}}$, are presented.

$\phi_{\text{sync-3a}}$				
Domains Λ				
Label	Regular language			
$\Lambda^{\#}$	$\Lambda_0^{\#} = (01)^*$, $\Lambda_1^{\#} = (10)^*$			
Particles P				
Label	Boundary	p	d	v
α	$\Lambda_0^{\#}\Lambda_1^{\#}, \Lambda_1^{\#}\Lambda_0^{\#}$	1	-1	-1
β	$\Lambda_0^{\#}\Lambda_1^{\#}, \Lambda_1^{\#}\Lambda_0^{\#}$	2	2	1
γ	$\Lambda_0^{\#}\Lambda_1^{\#}, \Lambda_1^{\#}\Lambda_0^{\#}$	5	-1	-1/5
Interactions I				
$\beta + \alpha \rightarrow \emptyset$		$\gamma + \alpha \rightarrow \emptyset$		
$\beta + \gamma \rightarrow \emptyset$				

$\phi_{\text{sync-3b}}$				
Domains Λ				
Label	Regular language			
$\Lambda^{\#}$	$\Lambda_0^{\#} = (01)^*$, $\Lambda_1^{\#} = (10)^*$			
Λ^d	$(111110)^*$			
Particles P				
Label	Boundary	p	d	v
α	$\Lambda^{\#}\Lambda^d$	2	0	0
β	$\Lambda^d\Lambda^{\#}$	5	3	3/5
γ	$\Lambda^{\#}\Lambda^d$	4	12	3
δ	$\Lambda^d\Lambda^{\#}$	1	-3	-3
ε	$\Lambda^d\Lambda^{\#}$	1	-3	-3
Interactions I				
$\beta + \alpha \overset{0.5}{\rightarrow} \delta + \gamma$		$\beta + \alpha \overset{0.5}{\rightarrow} \varepsilon + \gamma$		
$\alpha + \delta \rightarrow \emptyset$		$\gamma + \beta \rightarrow \emptyset$		
$\alpha + \varepsilon \rightarrow \emptyset$		$\gamma + \delta \rightarrow \alpha + \beta$		
$\gamma + \varepsilon \rightarrow \alpha + \beta$				

Appendix B

The Update Transducer of $\phi_{\textbf{bl-exp}}$

The update transducer $T_{\phi_{\text{bl-exp}}}$ of $\phi_{\text{bl-exp}}$ is given below in the following format:

```
i [0|a]->j [1|b]->k
```

This notation means that, if currently in state `i`, when a `0` is read, write symbol `a` and move to state `j`, and if a `1` is read, write symbol `b` and move to state `k`. The symbol `>` in front of a state means it is a start state. A `*` means an accepting, or final, state.

```
>*0  [0|0]->0  [1|0]->1          *32 [0|0]->0  [1|0]->1
 *1  [0|0]->2  [1|0]->3          *33 [0|0]->2  [1|0]->3
 *2  [0|0]->4  [1|0]->5          *34 [0|0]->4  [1|0]->5
 *3  [0|0]->6  [1|0]->7          *35 [0|0]->6  [1|0]->7
 *4  [0|0]->8  [1|0]->9          *36 [0|0]->8  [1|0]->9
 *5  [0|0]->10 [1|0]->11         *37 [0|0]->10 [1|0]->11
 *6  [0|0]->12 [1|0]->13         *38 [0|0]->12 [1|0]->13
 *7  [0|0]->14 [1|1]->15         *39 [0|0]->14 [1|1]->15
 *8  [0|0]->16 [1|0]->17         *40 [0|0]->16 [1|0]->17
 *9  [0|0]->18 [1|0]->19         *41 [0|0]->18 [1|0]->19
 *10 [0|0]->20 [1|0]->21         *42 [0|0]->20 [1|0]->21
 *11 [0|0]->22 [1|0]->23         *43 [0|0]->22 [1|0]->23
 *12 [0|0]->24 [1|0]->25         *44 [0|0]->24 [1|0]->25
 *13 [0|0]->26 [1|0]->27         *45 [0|0]->26 [1|0]->27
 *14 [0|0]->28 [1|0]->29         *46 [0|0]->28 [1|0]->29
 *15 [0|0]->30 [1|1]->31         *47 [0|0]->30 [1|1]->31
 *16 [0|0]->32 [1|0]->33         *48 [0|0]->32 [1|0]->33
 *17 [0|0]->34 [1|0]->35         *49 [0|0]->34 [1|0]->35
 *18 [0|0]->36 [1|0]->37         *50 [0|0]->36 [1|0]->37
 *19 [0|0]->38 [1|0]->39         *51 [0|0]->38 [1|0]->39
 *20 [0|0]->40 [1|0]->41         *52 [0|0]->40 [1|0]->41
 *21 [0|0]->42 [1|0]->43         *53 [0|0]->42 [1|0]->43
 *22 [0|0]->44 [1|0]->45         *54 [0|0]->44 [1|0]->45
 *23 [0|0]->46 [1|1]->47         *55 [0|0]->46 [1|1]->47
 *24 [0|0]->48 [1|0]->49         *56 [0|1]->48 [1|1]->49
 *25 [0|0]->50 [1|0]->51         *57 [0|1]->50 [1|1]->51
 *26 [0|0]->52 [1|0]->53         *58 [0|1]->52 [1|1]->53
 *27 [0|0]->54 [1|0]->55         *59 [0|1]->54 [1|1]->55
 *28 [0|0]->56 [1|0]->57         *60 [0|1]->56 [1|1]->57
 *29 [0|0]->58 [1|0]->59         *61 [0|1]->58 [1|1]->59
 *30 [0|0]->60 [1|0]->61         *62 [0|1]->60 [1|1]->61
 *31 [0|1]->62 [1|1]->63         *63 [0|1]->62 [1|1]->63
```

Note the regularity in state transitions in the transducer. From a state `i`, on reading a `0` a transition to state `2i mod 64` is made, and on reading a `1` a transition to state `2i+1 mod 64` is made. The output symbols on the transitions are, of course, determined by the CA update rule $\phi_{\text{bl-exp}}$, which was given in section 6.1.

References

[ABIK96] D. Andre, F. H. Bennett III, and J. R. Koza. Evolution of intricate long-distance communication signals in cellular automata using genetic programming. In C. G. Langton and K. Shimohara, editors, *Artificial Life V*, pages 513–520. MIT Press, 1996.

[Bäc97] T. Bäck, editor. *Proceedings of the Seventh International Conference on Genetic Algorithms.* Morgan Kaufmann, 1997.

[BB91] R. K. Belew and L. B. Booker, editors. *Proceedings of the Fourth International Conference on Genetic Algorithms.* Morgan Kaufmann, 1991.

[BCG82] E. R. Berlekamp, J. H. Conway, and R. K. Guy. *Winning Ways for your Mathematical Plays.* Academic Press, 1982.

[BE87] L. J. Bain and M. Engelhardt. *Introduction to Probability and Mathematical Statistics.* Duxbury Press, 1987.

[BH91] M. Boerlijst and P. Hogeweg. Self-structuring and selection: Spiral waves as a substrate for prebiotic evolution. In C. G. Langon, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Artifial Life II*, pages 255–276. Addison-Wesley, 1991.

[BJP+95] P. Borckmans, O. Jensen, V. O. Pannbacker, E. Mosekilde, G. Dewel, and A. De Wit. Localized Turing and Turing-Hopf patterns. In E. Mosek-

ilde and O. G. Mouritsen, editors, *Modelling the Dynamics of Biological Systems: Nonlinear Phenomena and Pattern Formation*. Springer, 1995.

[BM96] T. A. Brown and M. D. McBurnett. The emergence of political elites. In M. Coombs and M. Sulcoski, editors, *Proceedings of the International Workshop on Control Mechanisms for Complex Systems*, pages 143–161, 1996.

[BNR91] N. Boccara, J. Nasser, and M. Roger. Particlelike structures and their interactions in spatiotemporal patterns generated by one-dimensional deterministic cellular-automaton rules. *Physical Review A*, 44(2):866–875, 1991.

[Bon98] E. Bonabeau. Social insect colonies as complex adaptive systems. *Ecosystems*, 1(5):437–443, 1998.

[Bur70] A. W. Burks, editor. *Essays on Cellular Automata*. University of Illinois Press, 1970.

[CCNC97] P. P. Chaudhuri, D. R. Chowdhury, S. Nandi, and S. Chattopadhyay. *Additive Cellular Automata: Theory and Applications*, volume 1. IEEE Computer Society Press, 1997.

[CH93] J. P. Crutchfield and J. E. Hanson. Turbulent pattern bases for cellular automata. *Physica D*, 69:279–301, 1993.

[CIHY90a] K. Culik II, L. P. Hurd, and S. Yu. Computation theoretic aspects of cellular automata. *Physica D*, 45:357–378, 1990.

[CIHY90b] K. Culik II, L. P. Hurd, and S. Yu. Formal languages and global cellular automata behavior. *Physica D*, 45:396–403, 1990.

[CL89] J. Carroll and D. Long. *Theory of Finite Automata*. Prentice Hall, 1989.

[CM95] J. P. Crutchfield and M. Mitchell. The evolution of emergent computation. *Proceedings of the National Academy of Sciences, USA 92*, 23:10742–10746, 1995.

[CMD98] J. P. Crutchfield, M. Mitchell, and R. Das. The evolutionary design of collective computation in cellular automata. *Machine Learning Journal*, 1998. Submitted.

[Coo00] M. Cook. Universality in elementary cellular automata. In D. Griffeath and C. Moore, editors, *New Constructions in Cellular Automata*. Oxford University Press, 2000. In press.

[CPM95] P. E. Cladis and P. Palffy-Muhoray, editors. *Spatio-Temporal Patterns in Nonequilibrium Complex Systems*. Addison-Wesley, 1995.

[Cru94a] J. P. Crutchfield. The calculi of emergence: Computation, dynamics, and induction. *Physica D*, 75:11–54, 1994.

[Cru94b] J. P. Crutchfield. Is anything ever new? Considering emergence. In G. A. Cowan, D. Pines, and D. Meltzer, editors, *Complexity: Metaphors, Models, and Reality*, pages 515–533. Addison-Wesley, 1994.

[CY89] J. P. Crutchfield and K. Young. Inferring statistical complexity. *Physical Review Letters*, 63:105–108, 1989.

[Das98] R. Das. *The Evolution of Emergent Computation in Cellular Automata*. PhD thesis, Colorado State University, Fort Collins, CO, 1998.

[Dav81] H. A. David. *Order Statistics*. John Wiley & Sons, 1981.

[Dav91] L. D. Davis. *The Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.

[DCMH95] R. Das, J. P. Crutchfield, M. Mitchell, and J. E. Hanson. Evolving globally synchronized cellular automata. In L. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 336–343. Morgan Kaufmann, 1995.

[Dev89a] R. L. Devaney. *An Introduction to Chaotic Dynamical Systems*. Addison-Wesley, 2nd edition, 1989.

[Dev89b] P. Devreotes. Dyctyostelium discoideum: A model system for cell-cell interactions in development. *Science*, 245:1054–1058, 1989.

[DG89] J-L. Deneubourg and S. Goss. Collective patterns and decision making. *Ethology, Ecology and Evolution*, 1:295–311, 1989.

[DMC94] R. Das, M. Mitchell, and J. P. Crutchfield. A genetic algorithm discovers particle-based computation in cellular automata. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Parallel Problem Solving from Nature—PPSN III*, pages 244–353. Springer-Verlag, 1994.

[EEK93] G. B. Ermentrout and L. Edelstein-Keshet. Cellular automata approaches to biological modeling. *Journal of Theoretical Biology*, 160:97–133, 1993.

[Esh95] L. J. Eshelman, editor. *Proceedings of the Sixth International Conference on Genetic Algorithms*. Morgan Kaufmann, 1995.

[For90a] S. Forrest, editor. *Emergent Computation*. North-Holland, 1990. Special issue of Physica D, Volume 42, Nos. 1–3.

[For90b] S. Forrest. Emergent computation: Self-organizing, collective, and cooperative phenomena in natural and artificial computing networks. *Physica D*, 42:1–11, 1990.

[For93] S. Forrest, editor. *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann, 1993.

[FTW84] D. Farmer, T. Toffoli, and S. Wolfram, editors. *Cellular Automata*. North-Holland, 1984. Special issue of Physica D, Volume 10, Nos. 1–2.

[Fuk97] H. Fukś. Solution of the density classification problem with two cellular automata rules. *Physical Review E*, 55(3):R2081–R2084, 1997.

[Gar83] M. Gardner. *Wheels, Life and other Mathematical Amusements*. W. H. Freeman and Company, 1983.

[GBG+98] T. Gramss, S. Bornholdt, M. Gross, M. Mitchell, and T. Pellizzari. *Non-Standard Computation*. Wiley-VCH, 1998.

[GHR95] R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory*. Oxford University Press, 1995.

[GJ79] M. R. Garey and D. S Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[Gol89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[Gra94] C. M. Gray. Synchronous oscillations in neuronal systems: Mechanisms and functions. *Journal of Computational Neuroscience*, 1:11–38, 1994.

[Gre85] J. J. Grefenstette, editor. *Proceedings of the First International Conference on Genetic Algorithms*. Lawrence Erlbaum Associates, 1985.

[Gre87] J. J. Grefenstette, editor. *Proceedings of the Second International Conference on Genetic Algorithms*. Lawrence Erlbaum Associates, 1987.

[Gut91] H. Gutowitz, editor. *Cellular Automata: Theory and Experiment*. MIT Press, 1991. Special issue of Physica D, Volume 45, Nos. 1–3.

[Han93] J. E. Hanson. *Computational Mechanics of Cellular Automata*. PhD thesis, University of California, Berkeley, CA, 1993.

[HC92] J. E. Hanson and J. P. Crutchfield. The attractor-basin portrait of a cellular automaton. *Journal of Statistical Physics*, 66(5/6):1415–1462, 1992.

[HC97] J. E. Hanson and J. P. Crutchfield. Computational mechanics of cellular automata: An example. *Physica D*, 103:169–189, 1997.

[HCM98] W. Hordijk, J. P. Crutchfield, and M. Mitchell. Mechanisms of emergent computation in cellular automata. In A. E. Eiben, T. Bäck, M. Schoenauer, and H-P. Schwefel, editors, *Parallel Problem Solving from Nature—PPSN-V*, pages 613–622. Springer-Verlag, 1998.

[Hil84] W. D. Hillis. The connection machine: A computer architecture based on cellular automata. *Physica D*, 10:213–228, 1984.

[Hol75] J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975. Second edition: MIT Press, 1992.

[Hor97] W Hordijk. Correlation analysis of the synchronizing-CA landscape. *Physica D*, 107:255–264, 1997.

[HSC] W. Hordijk, C. R. Shalizi, and J. P. Crutchfield. An upper bound on particle interaction results in cellular automata. In preparation.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 1979.

[JP98a] H. Juillé and J. B. Pollack. Coevolutionary learning: A case study. In *Proceedings of the Fifteenth International Conference on Machine Learning*, 1998.

[JP98b] H. Juillé and J. B. Pollack. Coevolving the 'ideal' trainer: Application to the discovery of cellular automata rules. In *Proceedings of the Third Annual Genetic Programming Conference*, 1998.

[KS60] J. G. Kemeny and J. L. Snell. *Finite Markov Chains.* Van Nostrand Company, 1960.

[Lan84] C. G. Langton. Self-reproduction in cellular automata. *Physica D*, 10:135–144, 1984.

[Lan86] C. G. Langton. Studying artificial life with cellular automata. *Physica D*, 22:120–149, 1986.

[Lan90] C. G. Langton. Computation at the edge of chaos: Phase transitions and emergent computation. *Physica D*, 42:12–37, 1990.

[Lan91] C. G. Langton. Life at the edge of chaos. In C. G. Langon, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Artificial Life II*, pages 255–276. Addison-Wesley, 1991.

[LB95] M. Land and R. K. Belew. No perfect two-state cellular automata for density classification exists. *Physical Review Letters*, 74(25):5148–5150, 1995.

[LD94] G. Laurent and H. Davidowitz. Encoding of olfactory information with oscillating neural assemblies. *Science*, 265:1872–1875, 1994.

[Lip87] R. P. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, pages 4–22, April 1987.

[LN90] K. Lindgren and M. G. Nordahl. Universal computation in simple one-dimensional cellular automata. *Complex Systems*, 4:299–318, 1990.

[Man90] P. Manneville. *Dissipative Structures and Weak Turbulence.* Academic Press, 1990.

[MBVB90] P. Manneville, N. Boccara, G. Y. Vichniac, and R. Bidaux. *Cellular Automata and Modeling of Complex Physical Systems.* Springer-Verlag, 1990. Volume 46 of Springer-Verlag proceedings in Physics.

[MCH94a] M. Mitchell, J. P. Crutchfield, and P. T. Hraber. Dynamics, computation, and the "edge of chaos": A re-examination. In G. A. Cowan, D. Pines, and D Meltzer, editors, *Complexity: Metaphors, Models, and Reality*, pages 497–513. Addison-Wesley, 1994.

[MCH94b] M Mitchell, J. P. Crutchfield, and P. T. Hraber. Evolving cellular automata to perform computations: Mechanisms and impediments. *Physica D*, 75:361–391, 1994.

[Mei98] H. Meinhardt. *The Algorithmic Beauty of Sea Shells*. Springer, 2nd edition, 1998.

[MHC93] M. Mitchell, P. T. Hraber, and J. P. Crutchfield. Revisiting the edge of chaos: Evolving cellular automata to perform computations. *Complex Systems*, 7:89–130, 1993.

[Mit96] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1996.

[Moo97] C. Moore. Quasi-linear cellular automata. *Physica D*, 103:100–132, 1997.

[Moo98] C. Moore. Predicting nonlinear cellular automata quickly by decomposing them into linear ones. *Physica D*, 111:27–41, 1998.

[Mor98] B. M. Moret. *The Theory of Computation*. Addison-Wesley, 1998.

[MOW84] O. Martin, A. M. Odlyzko, and S. Wolfram. Algebraic properties of cellular automata. *Communications in Mathematical Physics*, 93:219–258, 1984.

[MRH86] J. L. McClelland, D. E. Rumelhart, and G. E. Hinton. *The Appeal of Parallel Distributed Processing*, pages 3–44. MIT Press, 1986.

[MTV86] N. Margolus, T. Toffoli, and G. Vichniac. Cellular-automata supercomputers for fluid-dynamics modeling. *Physical Review Letters*, 56(16):1694–1696, 1986.

[NNS97] H. F. Nijhout, L. Nadel, and D. L. Stein, editors. *Pattern Formation in the Physical and Biological Sciences*. Addison-Wesley, 1997.

[Nor89] M. G. Nordahl. Formal languages and finite cellular automata. *Complex Systems*, 3:63–78, 1989.

[O'C94] T. O'Connor. Emergent properties. *American Philosophical Quarterly*, 31(2):91–104, 1994.

[Pac88] N. H. Packard. Adaptation toward the edge of chaos. In J. A. S. Kelso, A. J. Mandell, and M. F. Shlesinger, editors, *Dynamic Patterns in Complex Systems*, pages 293–301. World Scientific, 1988.

[Par97] J. Paredis. Coevolving cellular automata: Be aware of the red queen! In T. Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 393–400. Morgan Kaufmann, 1997.

[PST86] J. K. Park, K. Steiglitz, and W. P. Thurston. Soliton-like behavior in automata. *Physica D*, 19:423–432, 1986.

[RTWE98] J. Rinzel, D. Terman, X.-J. Wang, and B. Ermentrout. Propagating activity patterns in large-scale inhibitory neuronal networks. *Science*, 279:1351–1355, 1998.

[SC99] C. R. Shalizi and J. P. Crutchfield. Computational mechanics: Pattern and prediction, structure and simplicity. *Communications of Mathematical Physics*, 1999. Submitted.

[Sch89] J. D. Schaffer, editor. *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann, 1989.

[SHC] C. L. Sidman, W. Hordijk, and J. P. Crutchfield. Mutational analysis of evolved cellular automata. In preparation.

[Sip97] M. Sipper. *Evolution of Parallel Cellular Machines: The Cellular Programming Approach.* Springer, 1997.

[SKW88] K. Steiglitz, I. Kamal, and A. Watson. Embedding computation in one-dimensional automata by phase coding solitons. *IEEE Transactions on Computers*, 37(2):138–145, 1988.

[Smi72] A. R. Smith. Real-time language recognition by one-dimensional cellular automata. *Journal of Computer and System Sciences*, 6:233–253, 1972.

[SN98] P. M. Simon and K. Nagel. Simplified cellular automaton model for city traffic. *Physical Review E*, 58(2):1286–1295, 1998.

[SS94] R. K. Squier and K. Steiglitz. Programmable parallel arithmetic in cellular automata using a particle model. *Complex Systems*, 8:311–323, 1994.

[Ter94] V. Terrier. Language recognizable in real time by cellular automata. *Complex Systems*, 8:325–336, 1994.

[TH88] P. Tamayo and H. Hartman. Cellular automata, reaction-diffusion systems and the origin of life. In C. G. Langon, editor, *Artifial Life*, pages 105–124. Addison-Wesley, 1988.

[TK94] H. M. Taylor and S. Karlin. *An Introduction to Stochastic Modeling.* Academic Press, revised edition, 1994.

[TM87] T. Toffoli and M. Margolus. *Cellular Automata Machines.* MIT Press, 1987.

[Tof84] T. Toffoli. Cellular automata as an alternative to (rather than an approximation of) differential equations in modeling physics. *Physica D*, 10:117–127, 1984.

[Vic84] G. Y. Vichniac. Simulating physics with cellular automata. *Physica D*, 10:96–116, 1984.

[vN66] J. von Neumann. *Theory of Self-Reproducing Automata.* University of Illinois Press, 1966. (Completed and edited by A. W. Burks).

[Win87] A. T. Winfree. *When Time Breaks Down.* Princeton University Press, 1987.

[Win90] A. T. Winfree. *The Geometry of Biological Time.* Springer-Verlag, 1990.

[WMC99] J. Werfel, M. Mitchell, and J. P. Crutchfield. Resource sharing and coevolution in evolving cellular automata. *IEEE Transactions on Evolutionary Computation*, 1999. Submitted.

[Wol83] S. Wolfram. Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55:601–644, 1983.

[Wol84a] S. Wolfram. Computation theory of cellular automata. *Communications in Mathematical Physics*, 96:15–57, 1984.

[Wol84b] S. Wolfram. Universality and complexity in cellular automata. *Physica D*, 10:1–35, 1984.

[Wol85] S. Wolfram. Twenty problems in the theory of cellular automata. *Physica Scripta*, T9:170–183, 1985.

[Wol94] S. Wolfram. *Cellular automata and complexity.* Addison-Wesley, 1994.