OPAALS

Contract N^{o} IST-034824

WP1:
Cell Biology, Autopoiesis and Biological Design Patterns

D1.4:
Mathematical Models of Gene Expression Computing

Project funded by the European Community under the "Information Society Technology" Programme.

Contract Number: IST-034824
Project Acronym: OPAALS

Deliverable No: D1.4
Due date: 31/05/2010
Delivery date: 11/10/2010

Short Description:
This report further develops the framework for linking biological behaviour to software behaviour specification, at an abstract level, and reports on the latest exciting experimental findings on the p53-mdm2 regulatory pathway. We argue that symbolic dynamics and algebraic automata theory are essential elements of this link. We discuss these two fields in some detail and show their relevance to the p53-mdm2 system. In particular, we show that it exhibits homoclinic behaviour, and is therefore amenable to a horseshoe map-type of analysis through symbolic dynamics, and that even at a fairly coarse discretisation the automaton derived from it harbours a simple non-abelian group in its holonomy decomposition.

Authors: Paolo Dini (LSE); Attila Egri-Nagy, Chrystopher Nehaniv, and Maria Schilstra (UH); Ingeborg Van Leeuwen, Alastair Munro, and Sonia Lain (UNIVDUN)
Partners contributed:
Made available to: Public distribution

Version	Date	Author, organisation
1	30/06/10	Dini (LSE), Egri-Nagy (UH)
2	31/08/10	Van Leeuwen (UNIVDUN)
3	15/09/10	Egri-Nagy (UH)
4	06/10/10	Dini (LSE), Egri-Nagy, Nehaniv, Schilstra (UH), Van Leeuwen (UNIVDUN)
5	11/10/10	Dini (LSE), Nehaniv (UH)

Quality Check:
1st Internal Reviewer: Sotiris Moschoyiannis (Surrey)
2nd Internal Reviewer: Paul Krause (Surrey)

Dependencies

Achievements*	Done: Aligned our epistemological perspective on interaction computing with the meta-theoretical framework developed in D12.10. Advanced experimental understanding of p53-mdm2 system. Greatly extended the computational power of the SgpDec semigroup decomposition computational algebra program. Discovered a simple non-abelian group in the automaton derived from the p53-mdm2 system. Identified oscillatory behaviour of p53-mdm2 system as homoclinic. Strengthened the plausibility of the theoretical framework for interaction computing based on category theory transformations and adjunctions, postulating the link between structure and behaviour to be related to the link between a Krohn-Rhodes and a symbolic dynamics view of finite-state automata, respectively. Not done: Did not have time to study symbolic dynamics to a sufficient depth to reach concrete conclusion about its links with algebraic automata theory. Did not, therefore, attempt to define a mathematical model for interaction computing. Although the plausibility of the approach has been strengthened, a few more years of work are likely to be needed before conclusive mathematical results in this area are reached.
Workpackages	WP1
Partners	LSE, UH, UNIVDUN
Domains	Interaction computing, automata theory, experimental cell biology, symbolic dynamics
Targets	WP1, WP3, interaction computing

Publications*	• A Egri-Nagy, P Dini, C L Nehaniv, and M J Schilstra. Transformation Semigroups as Constructive Dynamical Spaces. In Fernando A B Colugnati, Lia C R Lopes, and Saulo F A Barretto, editors, Digital Ecosystems: Proceedings of the 3rd International Conference, OPAALS 2010, pages 245265, Araca ju, Sergipe, Brazil, 22-23 March, 2010. Springer LNICST. • A. Egri-Nagy and C. L. Nehaniv. On straight words and minimal permutators in nite transformation semigroups. LNCS Lecture Notes in Computer Science, 2010. accepted. • P Dini and D Schreckling. A Research Framework for Interaction Computing. In Fernando A B Colugnati, Lia C R Lopes, and Saulo F A Barretto, editors, Digital Ecosystems: Proceedings of the 3rd International Conference, OPAALS 2010, pages 224244, Araca ju, Sergipe, Brazil, 22-23 March, 2010. Springer LNICST. • Attila Egri-Nagy and Chrystopher L. Nehaniv. Subgroup chains and Lagrange coordinatizations of nite permutation groups. arXiv:0911.5433v1 [math.GR], 2009. • Attila Egri-Nagy and Chrystopher L. Nehaniv. SgpDec software package for hierarchical coordinatization of groups and semigroups, implemented in the GAP computer algebra system, Version 0.5.19, 2010. http://sgpdec.sf.net. • G Horvath and P Dini. Lie Group Analysis of p53-mdm3 Pathway. In Fernando A B Colugnati, Lia C R Lopes, and Saulo F A Barretto, editors, Digital Ecosystems: Proceedings of the 3rd International Conference, OPAALS 2010, pages 285304, Araca ju, Sergipe, Brazil, 22-23 March, 2010. Springer LNICST. • S Lain, J J Hollick, J Campbell, O D Staples, M Higgins, M Aoubala, A McCarthy, V Appleyard, K E Murray, L Baker, A Thompson, J Mathers, S J Holland, M J R Stark, G Pass, J Woods, D P Lane, and N J Westwood. Discovery, in Vivo activity, and mechanism of action of a small-molecule p53 activator. Cancer Cel l, 13:454463, 2008. • I Van Leeuwen, A J Munro, I Sanders, O Staples, and S Lain. Numerical and Experimental Analysis of the p53-mdm2 Regulatory Pathway. In Fernando A B Colugnati, Lia C R Lopes, and Saulo F A Barretto, editors, Digital Ecosystems: Proceedings of the 3rd International Conference, OPAALS 2010, pages 266284, Araca ju, Sergipe, Brazil, 22-23 March, 2010. Springer LNICST. • I M M van Leeuwen and S Lain. Chapter 5: Sirtuins and p53. Adv Cancer Res, 102:171195, 2009.
PhD Students*	(none)
Outstanding features*	`SgpDec` computer algebra package
Disciplinary domains of authors*	Dini: applied mathematics, physics, computer science, social science Egri-Nagy: computer science, mathematics Nehaniv: mathematics, computer science Schilstra: biochemistry, computer science Van Leeuwen: mathematical, numerical, & experimental cell biology Munro: radiation oncology, biology Lain: cell biology, cancer drug development

* Indicates information requested by reviewers

Contents

Executive Summary

This report is the culmination of seven years of work across three different projects (DBE, BIONETS, and OPAALS) on the development of a mathematical theory of interaction computing. The objective, all along, has been to develop a mathematical bridge between cell metabolism and software systems, that would enable the latter to exhibit an analogous self-organising behaviour. However, all along we were keenly aware that the self-organising behaviour of cell metabolic systems is far from understood. Therefore, the research effort has actually been about solving *two* problems: an explanatory theory of systems biology **and** a new gene expression-inspired model of computation that would enable software systems to respond appropriately to external stimuli without having been explicitly programmed to do so in advance. Combining these two extremely challenging research problems has been a very interesting and successful epistemological experiment, which has led to a significant amount of joint theory construction both mathematically and in the interpretation of experimental results that would not have been possible by either field working in isolation.

The report focuses on three topics: symbolic dynamics, experimental cell biology, and algebraic automata theory. The basic concepts of symbolic dynamics are first summarised and discussed from the point of view of dynamical systems (which originally motivated the development of the field); a considerable effort is then expended in presenting the basic concepts of symbolic dynamics also from the point of view of coding. The motivation for the latter is that certain types of shift spaces can be understood as bi-infinite walks on finite-state automata, and therefore correspond to regular languages. This field therefore provides a link between formal languages, automata, coding, and dynamical systems within the same mathematical formalism. We did not go as far as attempting to develop a model of interacting automata using this approach.

We applied symbolic dynamics and algebraic automata theory work to the p53-mdm2 regulatory pathway in order to continue developing the dialogue between the mathematicians, computer scientists, and systems biologists from UH and LSE and the experimental cell biologists from UNIVDUN. The latter group reports on recent insights that led to the discovery of a new potential class of cancer drugs, the Tenovins. This work will form the basis for future mathematical work and model development.

We then spend a chapter on algebraic automata theory. We provide a broadly accessible introduction to the field intended for an interdisciplinary audience, explain the work performed by UH during the final year of OPAALS in developing an open source, very powerful computational algebra program for the holonomy decomposition of transformation semigroups, SgpDec, as a package of the open source GAP (Groups, Algorithms, Programming) computational algebra program. We apply SgpDec to two different automata, at two different levels of resolution, derived from the same p53-mdm2 system being investigated experimentally by UNIVDUN. The analysis shows that this system, at the higher resolution, harbours 2 simple non-abelian groups (SNAGs). SNAGs can be used as alternatives to Boolean algebra as functionally complete algebras as the basis for computation.

Finally, in the concluding remarks, we highlight how the concept of abstract number system could help unify symbolic dynamics with algebraic automata theory. Because symbolic dynamics provides analytical tools for studying the algebraic and dynamical properties of formal languages, whereas algebraic automata theory provides the analytical tools for studying the structural properties of automata, we argue that connecting these two complementary views of computing systems through a mathematical relationship could provide the constraint (adjunction) needed to solve the inverse problem of deriving automatically the structure that realises a given behavioural specification.

1 Introduction

1.1 An apology for Rationalism

The thread of research on interaction computing pursued in the DBE, BIONETS, and OPAALS projects [13, 14, 18, 15, 20, 22, 84, 16, 56, 83, 21, 19, 89, 51, 24], of which this report is the most recent representative, has espoused a point of view which appears to be firmly aligned with Descartes's causality and rationalism. In the course of the bio-computing research performed in these same projects, the evolutionary perspective has come to be associated with bottom-up and context-dependent order construction processes, in which the mutual process of discovery and adaptation between users and technology acquires a certain 'existential' flavour. Thus, in light of the 'existential pleasures'[1] of context-dependent evolutionary theory and of the inescapable influence on much of computer science and software engineering of bottom-up social constructivist processes, to hang on to Platonic Essentialism and Cartesian Determinism may seem a little 'out of touch', to say the least. However, as long as we remember that this research aims to uncover merely the *structural*, i.e. 'skeletal', characteristics of the dynamics of self-organisation (in Kauffman's sense [52]), leaving the 'flesh' and other context-dependent 'details' to be worked out through complementary ontological, epistemological, and methodological perspectives, then our call for an essentialist approach might be forgiven. For this brief discussion to provide a useful context for presenting the interaction computing rationale, it is helpful to recognise that the meaning of several philosophical concepts, such as Rationalism, has undergone significant changes in different historical periods since the time of Sir Francis Bacon (1600 AD).

Over the past century, there has been a growing trend to criticise so-called 'mechanistic' thinking, for a range of reasons whose breadth reflects the correspondingly various disciplines in which determinism, causality, and rationalism have come under scrutiny. While these concerns are warranted in many respects, there are different ways in which rationalism can be interpreted. It is helpful to recount the original understanding of the concept. Then, since it would take us too far afield to discuss its range of validity in the different contexts in which its adoption has been attempted, and more recently criticised, we will limit ourselves to show how a modern understanding that has received fairly wide acceptance, Karl Popper's Critical Rationalism, is not only plausible but actually necessary for the purposes of the interaction computing research agenda. We will use essentialism as the central reference concept in this discussion, which owes a great deal to Martin Hollis [49].

The ultimate essentialist theory is the mathematical study of symmetries, with which much of abstract algebra and in particular group theory is concerned, and of which engineering structures and patterns are the pragmatic expression. This duality between abstraction and concreteness is evocative of Hollis's account of causality in terms of two kinds of 'necessities'. The 17^{th} Century discussion about necessity concerned the fact that 'the senses reveal no necessities' ([49]: 34), meaning that generally by merely observing a physical phenomenon we cannot know why it must have occurred in that particular way.

About the physical or concrete form of causality Hollis says:

> Science is a search for causes, whereas observation cannot get beyond mere correlations. ... To explain an event is to identify its cause, thus placing it in a series of events each of which gives rise to the next. The series is not a mere sequence but one connected by the powers of the particulars involved to produce the next state in conformity with the laws of nature. ... Causes are thus ascribed some kind of necessity. (Ibid: 34)

[1] See [35] for an interesting and provocative digression.

Corresponding to the mathematical or abstract form of causality,

> Seventeenth-century rationalists ... were deeply impressed by the luminous qualities of mathematics, which they regarded as a model for scientific knowledge ... Mathematical truths have the interesting feature that they not only *are* true but *could not possibly be false*. ... Facts about numbers are objective and necessary facts of a universe which, at least in these ways, could not be otherwise. (Ibid: 35) [Emphasis in original]

Cartesian determinism has been rightly criticised for ascribing the qualities of the latter to the behaviour of the former. This is like saying that nature behaves in a particular way because the underlying (or, better, overarching) mathematics tells it so. According to Hollis, this conflation that seventeenth-century rationalists made of physical necessities with mathematical necessities was a mistake. Our view is that if, instead, we regard the mathematics as merely being concerned with developing *models* of physical behaviour that have a limited range of validity, then we leave nature the initiative and ascribe to mathematics merely the ability to describe a part of what nature does. This is the view that we follow in this report.

What is now called Cartesian determinism was one of the two possible ways in which Bacon understood the relationship between physical and mathematical necessities: the other way was from observables to general theories, which has given rise to Empiricism, the inductive method, and Positive Science. Whereas Bacon saw both progressions as 'rational', nowadays Positive Science is mainly associated with quantitative statistical methods for finding correlations between observables. Today, this is regarded as more *descriptive* than *explanatory*, meaning that statistical correlations and probabilities are not generally considered sufficient to explain *causal* relationships as hidden explanations of observed behaviour. As a consequence, Bacon's first way is now associated with rationalism, whereas his second way is associated with empiricism, and the two are seen as theoretically incompatible opposites. Figure 1 shows a diagrammatic view of these concepts. Although the Divine Watchmaker is a concept that comes from the early 19^{th} Century, we added it to this figure since its presence and role remained very much the same during the previous several centuries.

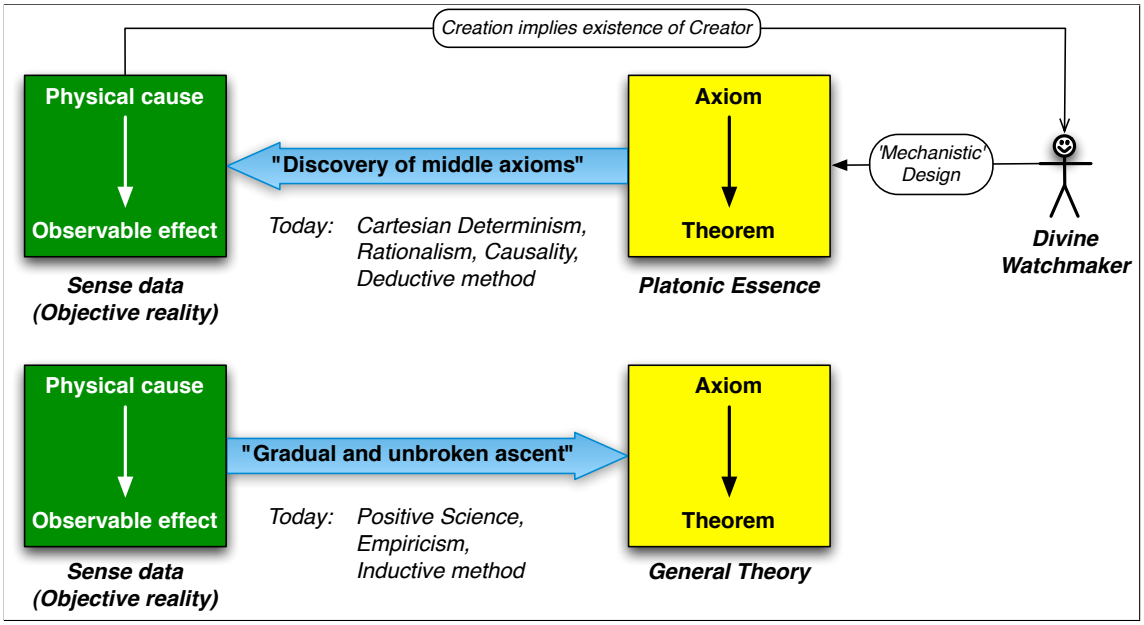


Figure 1: **Baconian Rationalism**

Although determinism goes too far in claiming that mathematical causality entails physical causality, of course logical deduction in mathematics *can* uncover relationships between parts of

a model that we may not have observed or even imagined between their physical counterparts. Therefore, mathematics *can*, and routinely does, uncover and 'explain' the unobservable. But no 'rational' modern scientist would dare believe any such predictions without some form of experimental verification (or absence of falsification [79]). This suggests that the meaning of rationalism has changed over time, and is now somewhat different to how it was understood by Bacon four centuries ago. Although a discussion of Kant would be appropriate at this point of our historical recap, since he proposed the first major synthesis of rationalism and empiricism, we fast-forward to modern times because in this report our discussion of certain aspects of the philosophy of science is meant to be merely in support of the very specific objective of developing a mathematical theory of Interaction Computing, and Popper's views will suffice.

As discussed more fully in deliverable D12.10 [17], Karl Popper has provided a 'rationalisation' that reflects well both modern scientific practice and the work performed in DE research, regardless of the discipline [79]. The Popperian framework could be seen as a workable integration of the deterministic, top-down, and deductive epistemology of rationalism with the positivist, bottom-up, and inductive epistemology of empiricism into a circular and never-ending iterative process of progressive refinement of models, as shown in Figure 2.

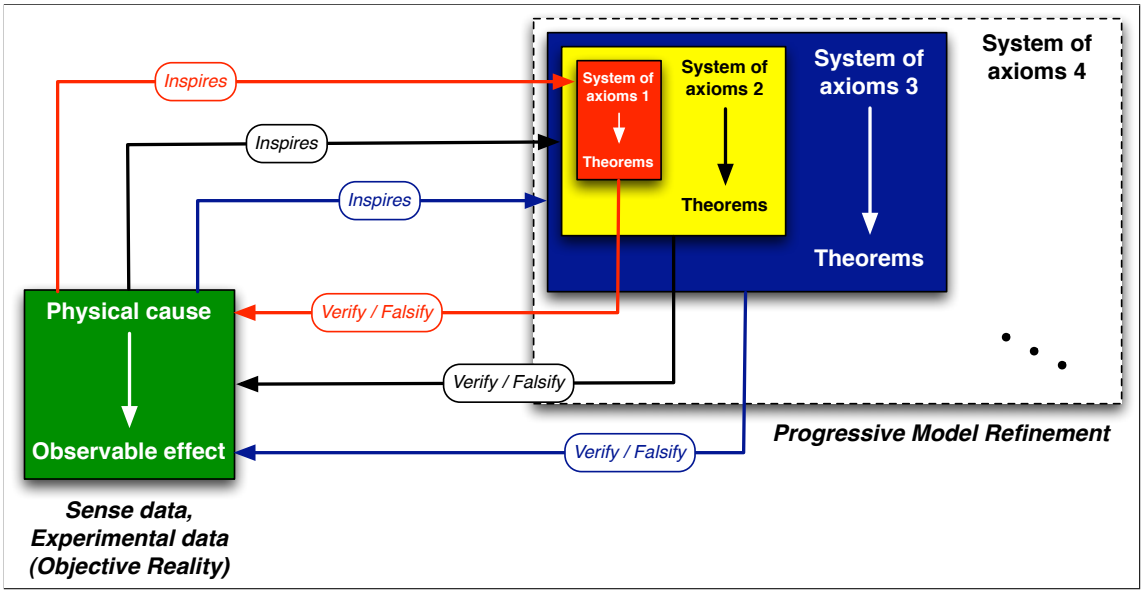


Figure 2: **Popperian (Critical) Rationalism**

Thus, the progressive model refinement espoused by Popper's rationalism is necessary because interaction computing attempts to replicate isomorphically the causal mechanisms underpinning order construction in cell metabolic and regulatory processes, in addition to relying on biological concepts as metaphors for new architectures and design patterns. This means that we wish to map the dynamical and structural properties of cellular processes to the dynamical and structural properties of computational processes. In particular, we wish to replicate in software the ability of the overall system to retain stability and robustness in spite of its openness to new components that enter its periphery, triggering a wide range of possible and unexpected interactions with its existing components. Since we are trying to approximate the behaviour of unimaginably complex processes and systems, we could not possibly hope to succeed in one go. Therefore, we must try to approximate the system whose properties we wish to emulate at some appropriately coarse initial level of abstraction, and improve our model gradually, as we understand more about the system and/or as we discover the fallacies of our initial approximations. The role of symmetry at the root of system stability to balance evolutionary open-endedness has already been discussed ([15]: 10; [17]) so we won't repeat the argument here.

1.2 Models in the natural/physical sciences and in computer science

A mention of models in a discussion of natural/physical and computer science could cause some confusion, because the concept of 'model' is treated differently in these two disciplinary domains. The most visible difference is that whereas mathematical models tend to be *analytical*, computer science models tend to be *synthetic*, as we now explain.

The models we have been discussing above are mathematical 'objects' that behave in some sense analogously, or even isomorphically at a given abstraction level, to a suitably chosen biological system (e.g. [89]). Examples of such models are systems of differential equations derived from the biochemical rate equations of a given metabolic or regulatory pathway [15, 16, 51]. Although the construction of such models is a creative and synthetic mathematical activity, they are meant to reproduce and help explain the behaviour of physical observables. Thus, their function is analytical.

It is interesting to note that during the course of the OPAALS project we have already experienced at least one iteration in the Popperian model refinement process. In fact, it has become apparent that the Lie group analysis of systems of non-linearly coupled ordinary differential equations (ODEs) modelling a given regulatory pathway such as the p53-mdm2 system [51] is probably too difficult and its applicability too limited for it to achieve useful predictions in a reasonable time (< 5 years). In addition, the ODE approach does not seem to be sufficiently granular, something we had already observed in previous reports (e.g. see [14]: 16). At the practical level of obtaining usable insights within a time-frame of a year or two, therefore, the Lie group approach has for the moment been 'falsified', prompting us to look elsewhere.

We had already begun to look at discrete models in three areas: symbolic dynamics was mentioned briefly ([14]: 16); network coding has been looked at in some detail in the BIONETS project [1]; and we began a study of abstract algebra in order to build a foundation of understanding to tackle algebraic automata theory [14, 15, 22, 16, 1]. Shifting our perspective from continuous to discrete models and extending the concept of model to encompass the semantics of both mathematical and computer science models will gain us several advantages:

- A clearer way to reconcile the different epistemological viewpoints, if not quite a single epistemology
- Further iterations in the Popperian model refinement process
- A mathematical object that can benefit from the *synthetic* or constructive activity of software engineering whilst remaining compatible with the *analytical* nature of mathematical models of biological behaviour – which we believe embody the constraints required to transfer biological behaviour to software.

1.2.1 Semantics

In the natural and physical sciences models, and in particular mathematical models, are able to reproduce the behaviour of observable variables because – to a suitable level of accuracy – they embody the relationships between them and the visible and invisible causes of their behaviour. For this reason they tend to be regarded as *explanatory*: the model provides a verifiable causal link between the observable effect and the invisible cause.[2] In software engineering, by contrast,

[2]For example, the velocity of a particle under the action of gravity can be derived from the mathematical 'model' $F = ma$, where F is indeed invisible.

'domain models' are descriptions of concepts relevant to a particular application domain and of the relationships between them. From these descriptions suitable data structures are derived, in support of functional code writing. Similarly, domain models can also serve as the basis for computational ontologies. Because such domain models account for all the causes and all the effects in a visible way, they tend to be called *descriptive* rather than explanatory.

However, if we include among the objects being described explicitly also the *causes* of observed behaviour, then also a mathematical model could be dubbed descriptive. In other words, a model that is explanatory at a certain level in a causal hierarchy could be considered descriptive at a higher level in that hierarchy. It could be argued, therefore, that mathematical models and domain models are not necessarily too dissimilar. Both are in essence descriptions of objects combined with the semantics of the relationships between them. Whereas the semantics of mathematical models are algebraic or calculus rules that reflect physical relationships, the semantics of domain models tend to be textual descriptions which obey specific grammar or syntax rules and reflect given functional and non-functional requirements. Both are formal systems and both can be made sufficiently self-consistent to allow, in the latter case, at least some level of 'reasoning'.

Therefore, the calculation of the consequences of mathematical models, such as the numerical evaluation of a system of differential equations to obtain the time variation of the concentration of a biochemical compound, are not too dissimilar to the deduction of logical consequences from a semantic network or ontology using the relationships explicitly labelled as links between its concepts. As formal systems go, mathematical formalisms tend to be better suited to model phenomena that ultimately arise from physical laws, possibly because with such models we have been trying to emulate physical behaviour for a long time, and the models had many chances to be falsified and subsequently further improved. The same mathematical formalisms, by contrast, are rather ineffective for describing and modelling software engineering artefacts. Domain models or other logic-based specification languages are much better suited. This perspective will be further explored in follow-up work inspired by [6].

1.2.2 Open model building

When computer science models need to embody biological behaviour, as in the case of bio-computing, clearly we need a new synthesis of these two perspectives. Thus, another important aspect of models is that, even if we were not aware of Popper's theories, the current state of the art in both mathematical and domain models can't help but be regarded as still primitive if we consider the expressiveness necessary to capture biological behaviour. We are therefore approaching the task of model building in each disciplinary domain, and the translations between them, with an open and critical mind: we should be ready to use ideas that have proven useful but without being fettered by *a priori* epistemological assumptions.

1.2.3 Structure

Although it appears to be in principle possible to place the semantics of the models representing biological and software systems on the same level, our claim is that it is not sufficient to do so if the objective is to arrive at software that behaves in a 'biological' way: we must take into account also the formal structure of the models. By 'formal structure' we mean the presence of a *specific* kind of structure, namely a *hierarchy* of nested formal structures charecterised by the fact that possibly different semantic rules apply at each level.

Structural hierarchies are important for several reasons:

- When hierarchical models are developed over time, their development usually starts with identifying the coarse skeleton mentioned above and thereafter tends to be consistent with the Popperian framework of progressive refinement.
- When increasing refinement corresponds to increasing specificity, model hierarchies can support multiple epistemological positions, i.e. the 'essentialist'/structuralist stability and generality of larger-scale structure along with the 'existentialist'/evolutionary context dependence of finer structure.
- They are implicitly present in most mathematical models, although not always visible, and have been researched for their cognitive function under the concept of 'coordinatisation': coordinatisation helps us understand complex problems [76, 77, 30, 81].
- They arise in software engineering artefacts simply as a consequence of architectural conceptualisation, requirements definition, and engineering design choices and optimisation.
- They are ubiquitous in biological systems.

We should clarify that in this discussion the term 'structure' is overloaded: it can carry the relatively more concrete meaning of the physical structure of biological systems and the structure of data or programs, but it can also refer to more abstract computational structures such as the algebraic structure of the transformation semigroups that formalise automata and similar computational abstractions.

At different scales biological systems are governed by different physical processes, which are modelled by correspondingly different mathematical models. Similarly, complex software applications can combine efficient device drivers written directly in Assembly at the lowest level, with objects or similar data structures at some intermediate computational level, with distributed online environments supporting multiple remote user interactions at the highest web execution level. Thus, structural hierarchies usually imply also functional hierarchies and hierarchies of models. As a consequence, in spite of the vast difference between biological and software systems the presence of some kind of structural hierarchy appears to be common to both.

1.2.4 Krohn-Rhodes theory

A theory that appears to be sufficiently general to support all of the above properties (semantics, openness of models, structure) is the Krohn-Rhodes theory for the decomposition of transformation semigroups [53],[3] which is why it has become the focus of attention of our research in interaction computing [15]. Confronted with the possibility that biological behaviour might only be possible if a stable set of elementary components, however many and however complex, is combined with a stable set of physical laws, we postulated that correspondingly universal structure laws might be needed to achieve self-organising behaviour in software. To achieve sufficient generality the manner in which we formalise structure must be abstract enough to be implementable in any applied context, and at potentially different scales. Further, it must naturally embody the dynamical properties of the systems it formalises ([21]: 114). This is how we have arrived at the conjecture that we are dealing with a structure imposed on a state space by a semigroup of state transformations and that therefore the appropriate object of study is the finite-state automaton. As explained in previous work [21, 24, 81], then, interacting finite-state automata appear to have the potential to model natural self-organising behaviour

[3] A transformation semigroup is a set of states together with a semigroup of transformations acting on it. Thus, it is the mathematical structure underpinning finite-state automata.

by formalising for software systems a suitable set of constraints derived from natural systems, and Krohn-Rhodes theory appears to provide a very interesting first step in this direction.

In [24] we discuss how transformation semigroups can be seen as constructive dynamical spaces, where algorithms are built by concatenating semigroup elements. This is in principle sufficiently expressive to account for program semantics; however, how a transformation semigroup can model the properties of the original dynamical system is still unclear. The open character of the modelling effort is not so much reflected by recourse to a finite-state automaton, whose expressiveness is limited relative to e.g. a Turing machine, but by the fact that the automata are allowed to interact [24, 81].

We hope eventually to develop testable hypotheses for the above claims. For the present, it seems that theorem proving is the more accessible verification method. For example, we will eventually want to prove whether or not mathematical models such as systems of ODEs can actually embody features and constraints that support self-organising behaviour. Or we will want to show that specific and recognisable algebraic structures are necessary and sufficient to formalise the constraints that support self-organising behaviour of suitably discretised metabolic systems. Or we will want to show that it is possible to translate such algebraic structures into specific kinds of algorithms and data structures. The concept of self-organisation, which we have currently described only conceptually as arising from the functional overloading of interacting finite-state automata, will need to be defined and expressed in a suitable formal system. The 'test' will then consist of building software environments in which software self-organisation as defined will result as a consequence of specific architectural specifications and dynamic qualities (i.e. algorithmic characteristics satisfying non-functional requirements). So far we have only been able to show that finite-state automata derived from cellular pathways do indeed have interesting algebraic properties. Chapter 4 of this report is therefore concerned with discussing the possible interpretations for this algebraic structure and with presenting the latest insights in the analysis, modelling, and emulation of biochemical systems and in particular of their ordered behaviour.

A rationale for discretising cellular pathways, usually going through a Petri net to obtain a finite-state automaton representation, has already been presented ([15]: 13; [21]: 114-115; [24]). Our recent and current research [77, 30, 26, 29, 32] shows several examples of cell regulatory and metabolic pathways to be formalisable as finite-state automata in this manner. The application of Krohn-Rhodes decomposition to the corresponding semigroups then reveals the presence of a rich algebraic structure in the form of permutation groups and non-invertible components (so-called flip-flops) at different levels of their hierarchical decomposition.

However, as we pointed out already ([15]: 15), the algebraic structure of automata does not account for their time-dependent or dynamical behaviour but, rather, for the space of computational trajectories they can carry out. In other words, although it accounts for all possible dynamical behaviours, in the sense of what the automata could possibly do, it does not provide timing information according to an external clocktime nor any information about relative likelihood of any of these trajectories.

As we discuss in Chapter 4, the holonomy decomposition of transformation semigroups yields groupings of automata *states* arranged in a hierarchy of nested sets of states, along with the transformations (elements of the semigroups) that operate on these sets of states.[4] The algebraic construction that formalises this idea is called the 'wreath product' (or a suitable

[4] 'Natural subsystems' and 'pools of reversibilty' (i.e. embedded permutation groups) are also identified by the decomposition, as will be discussed in Chapter 4.

substructure[5] thereof). This kind of structural analysis, however, does not tell us anything about the *algorithms* that might exhibit interesting or useful behaviour, and it does not tell how or why any such algorithms might be benefiting from the algebraic structure. Clearly once the holonomy decomposition is known many such algorithms can be derived or inferred, but they are not a systematic output of a Krohn-Rhodes style of analysis. This is why we felt we needed to complement the algebraic analysis of automata structure with an algebraic analysis of possible input strings, i.e. of the corresponding languages. This is addressed by the field of Symbolic Dynamics. This point needs to be explained carefully.

Applying 'forward engineering' to the automaton does give us all the possible input strings (which need not be restricted to inputs but could include internal events – not just 'inputs' – depending on the modelling). Automata constrain the possible dynamical trajectories. Formal language theory is very closely connected with automata and this relationship is very well studied, e.g. given a finite automaton one can give a regular expression for all the compatible input sequences, and conversely. So studying languages associated to automata is *not* a particularly good justification to move to the study of symbolic dynamics. But the timing information or likelihood of trajectories might be inferrable by symbolic dynamics methods. Another argument for symbolic dynamics may be as a *source* of languages and automata coming from dynamical systems to which we can apply algebraic analysis.

For example, given a particular family of dynamical behaviours, symbolic dynamics helps us determine whether this family defines a shift space (dynamical trajectory in phase space = point in the shift space). Once we do that, we may be able to infer the algebraic properties of such a shift space, and hence of the original dynamics. Such properties would then, hopefully, be transferrable to automata structure characteristics. This is true if the shift space is a sofic shift, for example. So the motivation to build up expertise in symbolic dynamics is the following possible scenario: we apply it to the analysis of complicated, non-linear dynamical systems that exhibit self-organising behaviour; if we are able to cast such a problem as two or more interacting systems, this ought to correspond to two or more shift spaces that are somehow 'communicating'; once we translate them into automata, we would have a way to derive a set of interacting automata along with all their structural characteristics. As we will discuss in the Conclusion, all this ought to be relatable to category theory adjunctions, although this is only speculation at this point. It is for these reasons that we feel symbolic dynamics deserves an in-depth look.

1.2.5 Symbolic dynamics

Symbolic dynamics originated in the work of Hadamard in discretising geodesic flows on surfaces of constant negative curvature [44]. A geodesic flow is nothing more than a generalisation to non-Eucledian spaces of the uniform (linear and constant) motion described by Newton's First Law in the absence of any forces on a particle or system. Symbolic dynamics grew as a tool for analysing general dynamical systems by discretising space [97], and it can therefore be seen as a complement to the Poincaré map, which transforms a continuous flow into a discrete iterated map by discretising time. As discussed by Morse and Hedlund,

> The methods used in the study of recurrence and transitivity frequently combine classical differential analysis with a more abstract symbolic analysis. This involves a characterization of the ordinary dynamical trajectory by an unending sequence of symbols termed a symbolic

[5] 'Substructure' is better than 'subset' here because it is actually a sub-transformation semigroup.

> trajectory such that the properties of recurrence and transitivity of the dynamical trajectory are reflected in analogous properties of its symbolic trajectory. [73]

Recurrence is the tendency of a dynamical system's trajectory to revisit a particular point or region of phase space, whereas transitivity is concerned with the ergodic properties of the trajectory, i.e. its tendency to come arbitrarily close to any point in a given phase space volume (generally defined by the system's energy).

Thus, from the beginning, symbolic dynamics was concerned with some aspects of the *global* and *long-time* properties of dynamical systems, expressed through an algebraic rather than a calculus formalism. As we will see in Chapter 2, this kind of analysis tells us which trajectories are admissible and which are not, where each trajectory is expressed as a doubly-infinite sequence of symbols from a given alphabet. Since we can define a special kind of space called a 'shift space' whose 'points' are these sequences, we can think of these infinite sequences of symbols as analogous to the infinite decimal expansion of a real number ([62]: xi). We can see, therefore, that symbolic dynamics works with mathematical objects that are compatible with continuous dynamical systems, but that are in fact composed of discrete entities. For this reason this theory seemed relevant to the bio-computing research agenda.

There are two more reasons that in our view justify investing in this area of research. First, one of the main areas of application of symbolic dynamics is network coding, which we have studied in the BIONETS project [20]. In particular, trellis codes [80] appear to be potentially relevant to the architecture of interacting automata. Second, as we will see in Chapter 2 certain kinds of shift spaces can be related directly to regular languages and finite-state automata.

Having provided a philosophical and mathematical overview underpinning interaction computing research, we now continue the development of the concepts related to interaction computing, with the understanding that this conceptual framework is only a stepping stone towards a suitable mathematical formalisation.

1.3 Interaction computing and its derivatives

We start by reproducing Figure 3 from D1.3 [15]. Whereas symbiotic computing is relevant to e.g. the concept of symbiotic security [83] and autopoietic computing is discussed in the companion deliverable D1.5 [7], this report is more concerned with gene expression computing.

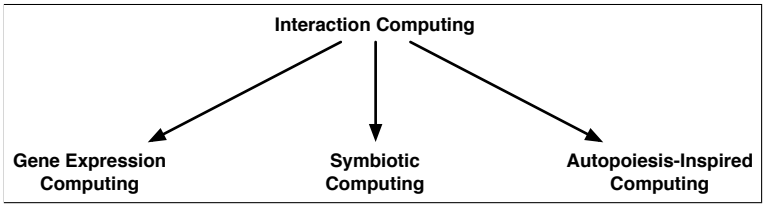


Figure 3: **Interaction computing and its derivatives**

1.3.1 Gene expression computing

Gene expression computing refers to the ability to specify environments that, in turn, can generate software services in response to internal or external stimuli. Thus, gene expression

computing aims to reproduce the ability of the biological cell to synthesise particular functional components such as enzymes in response to the needs of on-going metabolic processes or to signal transduction pathways from outside the cell. The signals to initiate the synthesis of such components are somehow 'encoded' in the metabolic processes themselves, i.e. in prior biochemical reaction products. If the purpose of these reaction products is to trigger the synthesis of the enzymes, then the causal chain is linear and relatively straightforward. If, instead, the reaction products are needed for some other purpose and, *at the same time*, they encode the trigger for the expression of the enzyme(s), then we have a case of functional overloading that is the root of the concept of symbiotic computing.

In any case, services generated in this manner should be considered 'atomic' in some suitable sense and many of them need to be composed dynamically to produce the complex behaviour that we associate with a software service of average size and complexity. The dynamic composition process is similar in character to the sequences of biochemical reactions that make up metabolic and regulatory pathways.

The above description glosses over an important point which we have not addressed properly in previous work and that we wish to rectify here. Since the first deliverable on bio-computing in the DBE project [13] we have acknowledged that software systems are devoid of interaction potential energy between their components as well as of a property analogous to temperature. The argument we have been making since then is that these two properties of physical and biochemical systems, upon which processes like the minimisation of free energy depend, are replaced in our vision of self-organising software ecosystems by user inputs that are propagated throughout the digital environment through a series of cascaded state transitions, similar to a domino effect. However, upon further scrutiny this does not now look realistic because, similarly to a biological system, the number of such external triggers is much much smaller than the number of internal components of the system and especially of the possible system states. Thus, it now seems more plausible to claim that the external triggers act like *signals* rather than as *actuators* of thermodynamic mixing.

The need to re-introduce an analogue to temperature is daunting but may not be such a foreign concept to software systems after all. From a statistical point of view, the effect of temperature in *physical* systems is to locate the system's internal dynamics at some point between total order (frozen structure of a crystalline solid) and total chaos (totally random and ever-varying structure of a gas). In a *biological* system, the energy implicit in a temperature value that falls in the lower half of the liquid phase of water is harnessed to drive a huge number of automatic functions that are nested and distributed at many scales. The signals that come from the 'outside' (a term whose precise meaning depends on the definition of 'system' being employed in any one instance) can only *modulate* the large number of operations that are already taking place. This execution architecture corresponds to a large number of 'Do Forever' loops that are interlocked and that keep a whole system running in some kind of 'idle' mode, like a large and dumb machine. This is where most of the 'energy' is. Low-energy external signals then behave like the base current in a transistor, they carry the critical *information* that can sway the large 'machine' in different directions. The functional overloading may then refer to the nesting, interdependence, and scalewise overloading of such control signals, a view that is strongly reminiscent of the cybernetics vision, and in particular of Ross Ashby's Law of Requisite Variety [2].

To postulate a huge number of idle loops cycling through a fixed number of states, at different scales, is not so unreasonable if we compare this picture to the properties of large digital systems. In the internet, for example, servers, switches, and routers are constantly performing an astronomical number of repetitive operations that at the lowest syntactical level are rather

meaningless but that are nonetheless compiled from higher-level instructions. The difference between the internet today and a biological system is that in the former the number of internal operations is of a similar order of magnitude as the number of external stimuli, whereas in the latter this is not the case. The number of internal operations is astronomically larger than the number of external stimuli.

What allows us to make this claim for biological systems is that we are neglecting molecular interactions below the threshold of the simplest organic molecules. This allows us to separate external stimuli in a similar way: for example, we can neglect the collision of air molecules with the skin, but consider smells as valid signals. In the case of the internet, we are separating the internet itself from the applications that may be running on servers and which obviously contain a huge number of operations. What makes this possible is the engineering practice of modularisation and layering, which separates different aspects of the overall systems. In biology, on the other hand, subsystems cannot *afford* to be independent because there would be nothing transmitting the *information* between them that underpins their cooperation or, perhaps better, choreography. In other words, in biology every subsystem is tightly coupled if not to everything else at least to its adjacent subsystems, whereas in software engineering we have been striving to work towards loose coupling as much as possible.

Thus, a necessary architectural requirement for self-organising systems built on gene expression computing appears to be to base high-level functions on a nested hierarchy of basic and repetitive functions that can be increasingly automated as one moves down the hierarchy towards smaller scales of system description and ever-greater numbers of components.

1.3.2 Symbiotic computing

At an applied level symbiotic computing can be explained through the example of symbiotic security. As discussed in Schreckling et al. ([83]: 36), symbiotic security captures the balance between the encapsulation of a security function in a specialised security service with the interdependence between such a service and the service(s) it is meant to secure. In other words, in order to address the problem posed by the 'ex post patching paradigm' in the development of security services and applications,[6] we felt it would be worth investigating a mode of generation of such paired services based on the integration of their most fundamental functions ex ante. This is similar to how the human immune system is inextricably integrated with the nervous and endocrine systems [11].

As a consequence, in symbiotic security a security service is assumed to be tightly coupled to the services it is meant to secure. Thus, neither can be executed by itself, both need the other service to do anything. Each service is functionally overloaded to perform its function *and* to trigger state transitions in the service it is coupled to. Since the concept of service applies at different scales, gene expression computing becomes a general framework that can support the more specific symbiotic computing, whose realisation requires the enforcement of additional suitable constraints.

[6] The ex post patching paradigm does not develop new security services or applications but, rather, patches the corresponding applications after realising that they have a problem.

1.3.3 Autopoietic computing

Autopoietic computing [7], finally, is concerned with adding recursion as a fundamental property of every element of the architecture. Thus, similarly to how we can say in a service-oriented architecture that 'everything is a service', so in autopoietic computing we can say that 'everything is an autopoietic service'. This means that every element of the architecture, at every scale, is overloaded, as follows:

- The primary function is defined by functional specifications as in any architecture.
- The secondary function that the primary function *also* contributes to is to support the replication of a strategically defined architectural unit, which is analogous to the biological cell.

It is not unreasonable to postulate that the manner in which the autopoietic function is realised may depend on applying recursion to symbiotic computing. In other words, a framework that could integrate all three types of computing might be to overload three or even more functions on cellular components:

- The primary function is defined by functional specifications as in any architecture.
- The secondary function that the primary function *also* contributes to is to support other metabolic pathways, for instance by triggering the expression of suitable enzymes.
- *(Other intermediate and probably scale-dependent functional levels are possible)*
- The highest-level function that the primary, secondary, etc functions *also* contribute to is ultimately to support the replication of a strategically defined architectural unit, which is analogous to the biological cell.

The hypothesis is that as we go from a single component to the whole cell (or its analogue in software) the different levels of functional overloading are coupled by binary symbiotic computing relationships.

The principle by which the highest-level function of multiple components is organised is called 'operational closure' and has been discussed in [14, 18, 15], as well as in [7] and in the original sources [65, 66]. The motivation for attempting to achieve the highest-level, replicative function is to increase the morphogenetic and adaptation properties of software applications and services through a continual renewal of its core components. The continual renewal of lower-level components could be one type of the 'Do Forever' loops that are necessary to hold the system in a dynamic state that can transition quickly to different 'macrostates' depending on the inputs it receives, as we discussed above. In other words, autopoietic computing is important for extending the self-organisation ability of single atomic services to greater scales, ultimately aiming for veritable autonomous 'digital organisms'. The difference is already well-established in computer science as the difference between a module containing some functional code and a module containing some functional code *and* an internal representation of itself and its environment – i.e. an *agent*. Surprisingly, to integrate the various types and levels of bio-computing it appears that such agents will need to be tightly rather than loosely coupled, which goes against current trends in the architecture of distributed software systems.

1.3.4 Integrating the different types of bio-computing

Once the interactions inside the cell, between cells, and between aggregates of cells have been translated into computer science rules subject to suitable algebraic constraints, we will be ready

to build an evolutionary framework around the whole construction and will finally be able to 'launch' a software ecosystem and watch it evolve over time. Figure 4 (based on a similar figure in [21]: 119) shows the relative importance of different modelling perspectives at different scales. Although the time axis is technically a 3rd dimension, in this 2D diagram it is drawn parallel to the length scale axis; a 3D diagram would have been too difficult to draw and not very clear.

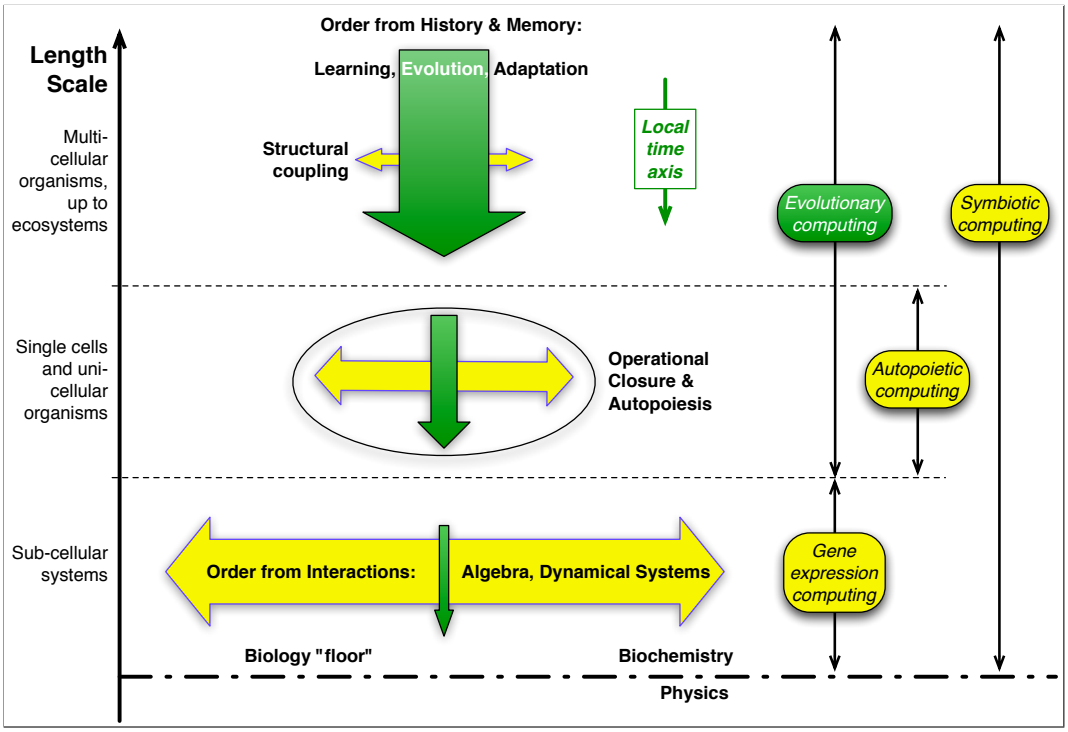


Figure 4: **Schematic showing the different relative importance of order construction mechanisms in biology, at different scales of description, and the corresponding relevance of different kinds of bio-computing. Green background indicates memory/history-dependent processes developing over time, yellow background indicates instantaneous dynamical interactions and reliance on a scale-invariant mathematical model of interaction computing.**

The argument so far sounds good, but unfortunately it rests on a clay foundation. Nothing of what has been described will be possible without resolving a root problem, which we see as a mathematical problem in its essence: we need to understand and formalise the (probably recursive and scale-invariant) mathematical nature of interaction computing. All our work to date suggests that this is a very difficult mathematical problem, but that, if solved, it has the potential to revolutionise how we think about computing and how we go about designing and implementing software.

In this final report on interaction computing in the OPAALS project we have not solved this mathematical problem. However, we feel we have made significant conceptual, theoretical, and mathematical advances. In Chapter 2 we give an introduction to symbolic dynamics to provide a basis for further work and exploration. In Chapter 3 we give an update on recent experimental findings in the p53-mdm2 system. In Chapter 4 we provide an update on our results in algebraic automata theory. Finally, in Chapter 5 we sum up the last 8 years of insights into a few highlights and draw some conclusions that will set the stage for our next steps in this line of research.

2 Symbolic Dynamics

In this chapter we need to introduce the basic concepts of symbolic dynamics in order to provide a foundation for further work and an eventual bridge to algebraic automata theory. We begin with a discussion of how symbolic dynamics has been applied to the long-time analysis of non-linear dynamical systems [47] in order then to show some striking similarities with the p53-mdm2 ODE model we developed in previous OPAALS work [15, 89]. Where needed we also rely on Williams's very clear explanations [97]. The second part of this chapter focusses instead on Lind and Marcus's textbook on symbolic dynamics and coding [62]. As in all our deliverables on bio-computing in the DBE, BIONETS, and OPAALS projects (e.g. [13, 22, 16, 14]), the detailed discussion of textbook material is done for two main reasons:

- Our discussion usually goes in more detail than the textbook it is drawn from whilst being selective of the topics that are discussed. This is done to make the material more accessible to a wider interdisciplinary audience while optimising the presentation for the objectives of the report.
- Since our research programme in bio-computing began in 2002 and is likely to continue for at least another 5 years, a very careful exposition is invaluable to guarantee *our own* ability to access and reconstruct the arguments months or years after the reports have been written.

In order to put the discussion of symbolic dynamics in context, Figure 5 shows how this topic fits within the broader mathematical research workflow that we have been following and documenting in several reports in the OPAALS and BIONETS projects. We meant to look for a possible bridge between the discretisation of the p53-mdm2 system by symbolic dynamics and the discretisation of the same system via the Petri net approach, discussed in [24] and in Chapter 4 in this same report, but we ran out of time. Therefore, the second part of this chapter will instead summarise the basics of symbolic dynamics as presented by Lind and Marcus [62] from a coding rather than a dynamical systems perspective. The motivation is that Lind and Marcus's text is comprehensive and covers also sofic systems in detail, i.e. shift spaces that correspond to finite-state automata. Although we did not get this far in the OPAALS project, our next step in the development of a suitable mathematical framework for interaction computing will be to reconcile Krohn-Rhodes theory with the symbolic dynamics perspective.

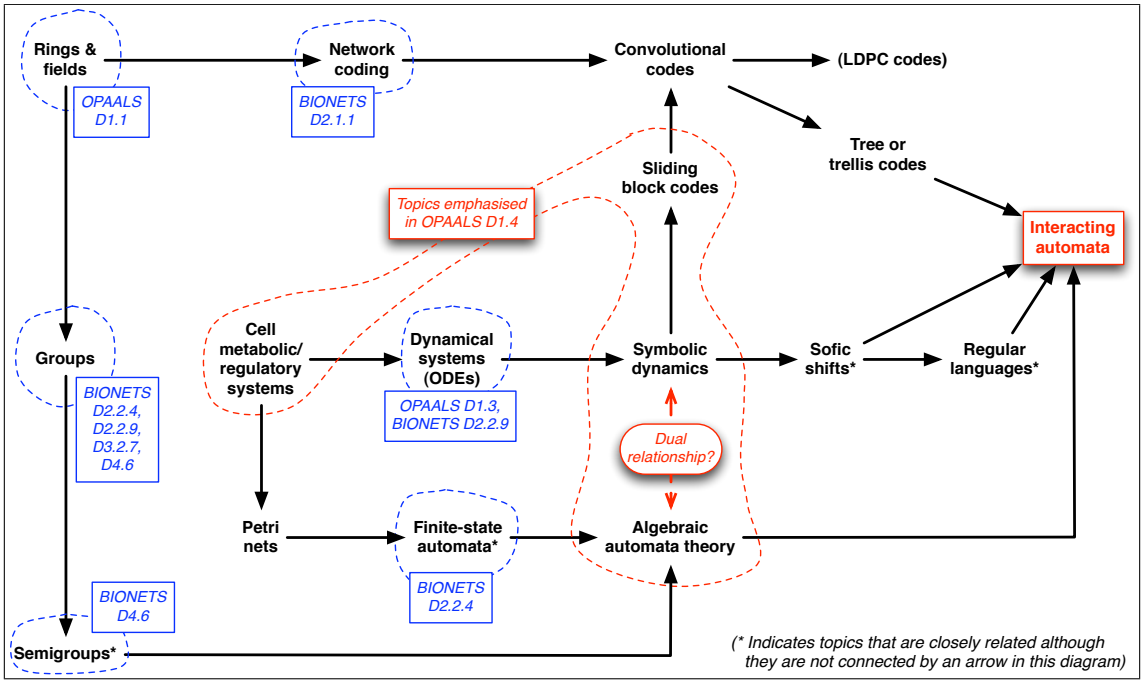


Figure 5: **(Tentative) high-level view of the mathematical research workflow**

2.1 Symbolic dynamics from the point of view of dynamical systems

2.1.1 The logistic map

In this section we build some basic machinery that will enable us to show how symbolic dynamics can be used to analyse some of the properties of chaotic dynamical systems. An introduction to the history of the logistic map as a deceptively simple model for population growth can be found in most introductory books on chaos theory, such as [47] itself. One of our objectives in this chapter is to maintain the link to dynamical systems as intuitively clear as possible. This is because as we develop the abstract formalism of symbolic dynamics and eventually reach shift spaces that are equivalent to finite-state automata what tends to happen is that one forgets the intuitive connection to dynamical systems and is therefore less able to apply physical intuition, for example, in interpreting abstract algebraic automata theory results (e.g. Chapter 4).

Thus, we start with the 1st-order ordinary differential equation for the logistic population growth model

$$x' = \lambda x(1 - x), \tag{1}$$

If we plot the derivative as a function of x, in the spirit of the jet space ([16]: 26; [51]), we get a graph as shown in Figure 6.[7] In this figure λ is approximately equal to 4.2.

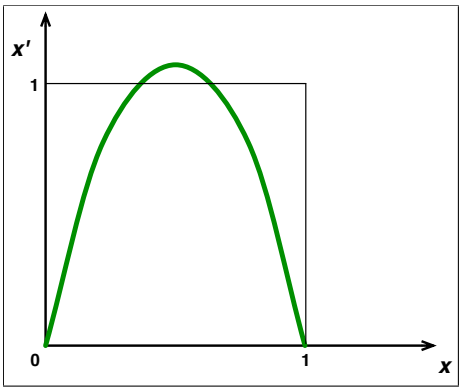


Figure 6: **Visualisation of the rate of change of x as a function of x**

The equation we want to work with is a discrete version of this system, and can be approximated somewhat crudely by the so-called logistic map, shown for a similar value of λ in Figure 7:

$$f_\lambda(x) = \lambda x(1 - x). \tag{2}$$

Here $f_\lambda(x)$ is the fraction of some optimal population level at a future time-step as a function of the current level. This is an iterated map, meaning that the result $f_\lambda(x)$ is fed back into the function as the argument x, and for any initial value of $x = x_0$ in the unit interval $I = [0, 1]$ this process is repeated forever. As shown in Figure 8, there is an easy graphical visualisation of the orbit. From any starting point x_0 in the unit interval one draws a line to $f_\lambda(x_0)$. This now becomes the next x, say x_1. If we draw a horizontal line through $f_\lambda(x_0)$, the point where it intercepts the diagonal $y = x$ will necessarily be above the value $x_1 = f_\lambda(x_0)$. Thus, from this point we just have to move up or down until we meet the logistic curve again, which is just $f_\lambda(x_1) = x_2$, and so forth. Figure 8a shows how all the points that start within the interval

[7] In [16, 51] the graph is a surface because the ODE being analysed is non-autonomous. The equation given here on the other hand is not a function of time, so a simple line graph is sufficient.

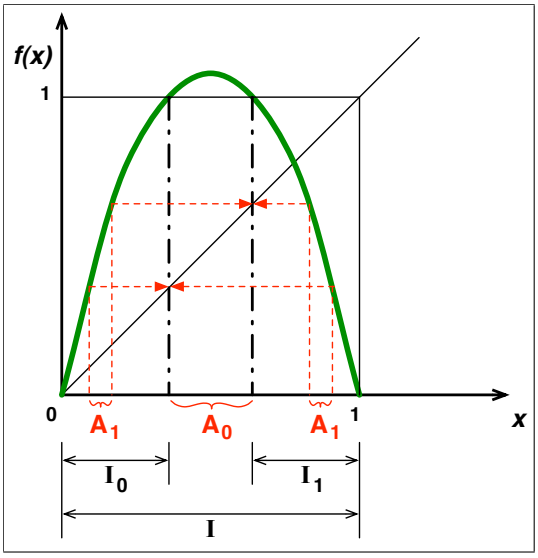


Figure 7: **Logistic map showing the discretising intervals (I_0, I_1), the region that escapes to infinity (A_0), and its first pre-image (A_1)**

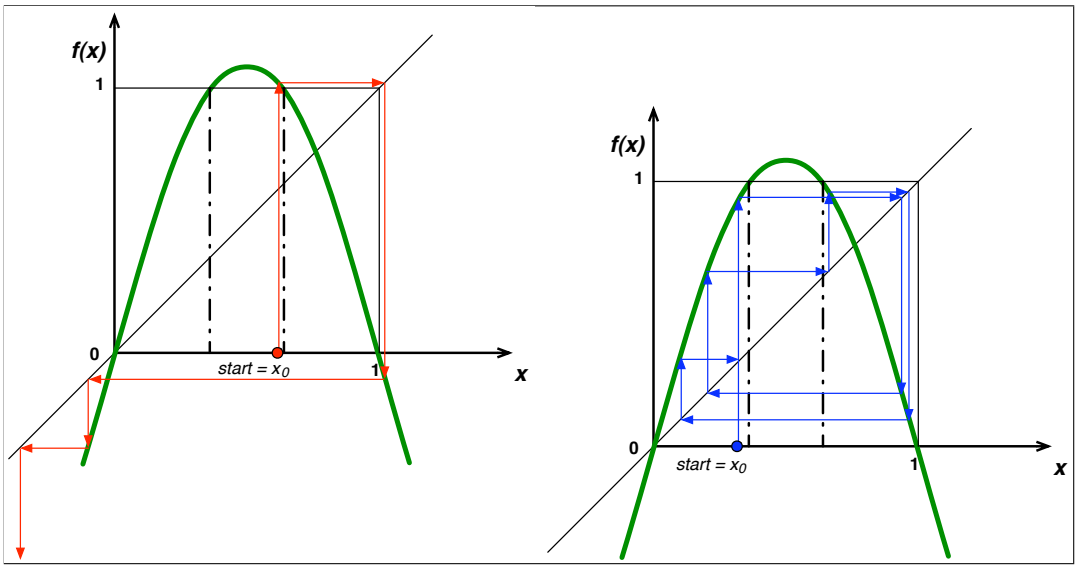


Figure 8: **Example of an escaping orbit and of a periodic orbit**

A_0 shown in Figure 7 will certainly escape to infinity, whereas the points that start in I_0 or I_1 may remain within the unit interval, such as Figure 8b. However, clearly the pre-images of A_0, the two segments labelled A_1 in Figure 7, will also escape, and so will their pre-images, and so forth. The disjoint set of points that do *not* escape is called Λ:

$$\Lambda = I - \bigcup_{n=0}^{\infty} A_n, \tag{3}$$

Now take $x_0 \in \Lambda$ and look at $f_\lambda^n(x_0)$. It is easy to see that the entire orbit of x_0 must lie in $I_0 \cup I_1$. Let $x_n = f_\lambda^n(x_0)$, and let $x_n = 0$ or 1 depending on whether $f_\lambda^n(x_0) \in I_0$ or I_1, respectively. Thus, for each point $x_0 \in \Lambda$ there exists an infinite sequence $S(x_0) = (s_1 s_2 s_3 \dots)$ of 0s and 1s called the **itinerary** of x_0.

Let $\Sigma = \{\text{All the possible infinite sequences from } \mathbb{Z}_2\}$. We can regard Σ as an abstract space,

and each infinite sequence as a 'point' in this space. We can define a distance over Σ between two points $s = (s_1 s_2 s_3 \dots)$ and $t = (t_1 t_2 t_3 \dots)$ as long as the definition satisfies the three required properties of equivalence, symmetry, and triangle inequality. A function that does is:

$$d(s,t) = \sum_{i=0}^{\infty} \frac{|s_i - t_i|}{2^i}. \tag{4}$$

Since the geometric series $\sum_{i=0}^{\infty} x^n$ converges to $\frac{1}{1-x}$ for $x < 1$,

$$\sum_{i=0}^{\infty} \frac{1}{2^i} = \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i = \frac{1}{1-1/2} = 2. \tag{5}$$

Therefore, our definition of distance also converges since the numerator for each value of i is either 0 or 1. With this, we can show

Proposition 2.1 If two sequences s and t are equal in their first $n+1$ values, then $d(s,t) \leq 1/2^n$.

Proof.

$$\begin{aligned} d(s,t) &= \sum_{i=0}^{n} \frac{|s_i - t_i|}{2^i} + \sum_{i=n+1}^{\infty} \frac{|s_i - t_i|}{2^i} \\ &= 0 + \frac{|s_{n+1} - t_{n+1}|}{2^{n+1}} + \frac{|s_{n+2} - t_{n+2}|}{2^{n+2}} + \dots \\ &= \frac{1}{2^{n+1}} \left(\frac{|s_{n+1} - t_{n+1}|}{2^0} + \frac{|s_{n+2} - t_{n+2}|}{2^1} + \dots \right) \\ &\leq \frac{1}{2^{n+1}} \sum_{i=0}^{\infty} \frac{1}{2^i} = \frac{1}{2^{n+1}} 2 = \frac{1}{2^n} \end{aligned} \tag{6}$$

□

We are going to state and explain but not prove the first important result of symbolic dynamics:

Theorem 2.2 The itinerary function $S : \Lambda \to \Sigma$ is a homeomorphism for $\lambda > 4$.[8]

This theorem is saying that for each point $x \in I$ that does not escape under the repeated action of $f_\lambda(x)$ there exists one and only one sequence $S(x) \in \Sigma$. Therefore, by studying the properties of the elements of Σ, i.e. infinite sequences of symbols taken from the alphabet $\mathbb{Z}_2$, we will automatically be studying the properties of the orbits of the logistic map.

Definition 2.3 The map $f_\lambda : I \to I$ is **chaotic** if

- Its periodic points are dense in I, i.e. they come arbitrarily close to any point in I.
- f_λ is transitive on I: for $U_1, U_2 \subset I, \exists n$: if $x_0 \in U_1, f_\lambda^n(x_0) \in U_2$.
- f_λ has sensitive dependence on I: this means that the distance between any two arbitrarily close points will diverge and will be greater than an arbitrarily chosen constant after a sufficient number of iterations. In mathematical language, this concept is expressed as follows: let $U \subset I$ be an arbitrariry small open subset of I centred around x_0. Then, $\exists \beta > 0 : \forall x_0 \in I$ and $x_0 \in U$, $\exists (y_0 \in U, n > 0) : |f_\lambda^n(x_0) - f_\lambda^n(y_0)| > \beta$.

Definition 2.4 Let I and J be two different intervals, and $f : I \to I$, $g : J \to J$. Then, f and g are **conjugate** if there exists a homeomorphism $h : I \to J$ such that $h \circ f = g \circ h$.

[8] A homeomorphism is a 1-1, onto, invertible, and continuous map.

2.1.2 Number expansions in different bases

Now let's look at another possible map, shown in Figure 9 along with a possible orbit, which is constructed as above. This is called the doubling map, $g : I \to I$:

$$g(x) = 2x \bmod 1, \tag{7}$$

and its dynamics are particularly interesting. In fact, if the unit interval is divided into I_0 and I_1 as shown and the symbols of the itinerary are taken from $\mathbb{Z}_2$, then the itinerary for any $x \in [0, 1]$ is none other than its binary expansion.

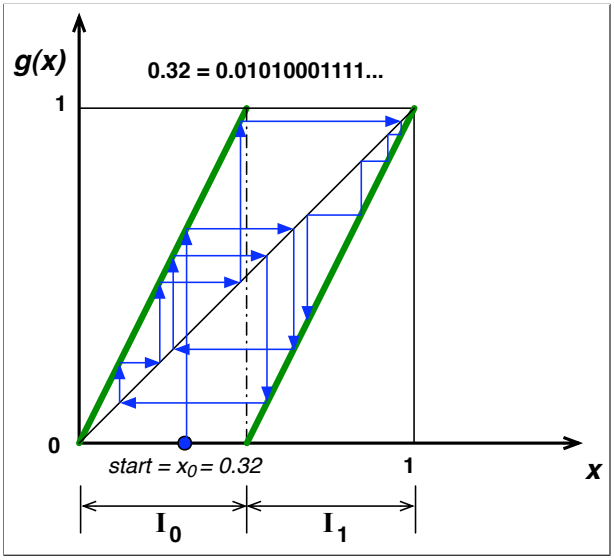


Figure 9: **Doubling map, showing the binary expansion of the number 0.32**

The binary expansion and doubling map may look unfamiliar, but they become more accessible if one compares them with the decimal equivalent (over the same interval), $h : I \to I$:

$$h(x) = 10x \bmod 1, \tag{8}$$

It should be fairly obvious that repeated iteration of the above map applied to a decimal real number in decimal notation will give that number back. Figure 10 shows a graphical visualisation of the decimal-to-binary conversion for fractional numbers, to aid the intuition.

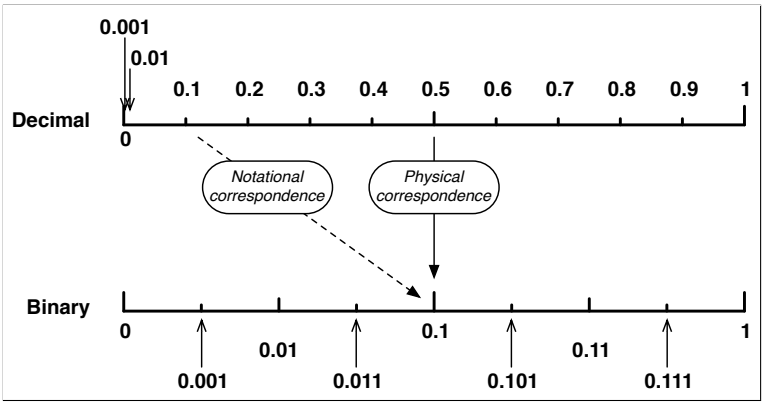


Figure 10: **Conversion between decimal and binary fractional numbers**

It would appear from the above that the concept of discrete iterated map is quite powerful, so that not only can it model dynamical systems like the population growth equation, but also conversions between different number systems. However, the converse of this statement is more interesting: the concept of number system appears to be so powerful that it can be generalised. In other words, the logistic map can be seen as a *very radical* generalisation of the concept of number system: it appears that symbolic dynamics gives a way to express the behaviour of a dynamical system by means of a discrete 'model' that in its essence is no different to a (rather surprisingly defined) number system.

The reason for pushing this somewhat strange point is that it is beginning to look like an insight of rather general relevance. In fact, it turns out that the automata decomposition theory and results that we will discuss in Chapter 4 can most easily be understood and conceptualised through the same identical concept of generalised number system.

There is a more abstract and at the same time more intuitive way to express this concept, as 'coordinatisation'. This has been discussed in many publications in the last 20 years in the specific context of Krohn-Rhodes theory [76, 77, 30, 81], as we already mentioned in the Introduction, but it can certainly be considered a concept that has pervaded the mathematical modelling of *any* and *all* physical phenomena for as long as mathematical modelling has existed as a cognitive process. It appears to be truly a concept of fundamental importance.

The leap between the binary expansion and the logistic map is perhaps too big to make in one step, even at an intuitive level. It may therefore help to mention the so-called β-expansions, i.e. expansions in a non-integer base [97, 36]. If for example we let $\beta = \frac{1+\sqrt{5}}{2} = \gamma$, the so-called 'golden mean', the map $g : I \to I$ in this case is:

$$g(x) = \gamma x \bmod 1, \qquad \gamma = \frac{1+\sqrt{5}}{2}, \tag{9}$$

and the corresponding graph is shown in Figure 11. To help understand this diagram, note how, because of the form of both the doubling map (7) and the golden mean map (9), the point where the function meets $g(x) = 1$ is 1/base. This, therefore, is the point that divides the two intervals I_0 and I_1, in both cases.

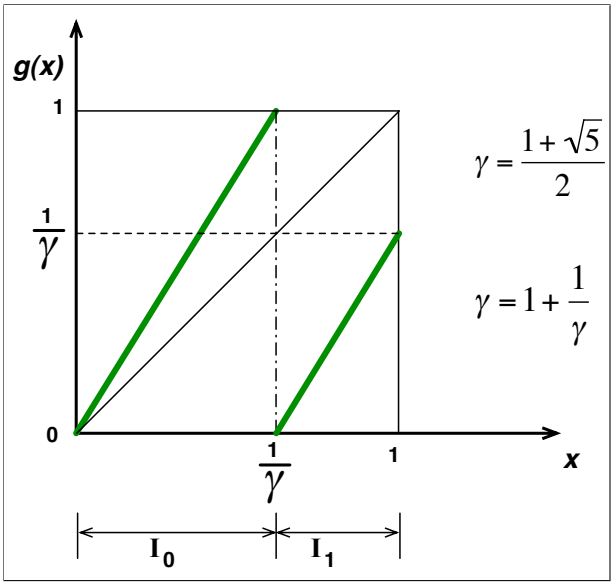


Figure 11: **Map for a non-integer β-expansion. In this case β = golden mean = γ**

It should be fairly easy to see from this graph that any trajectory that falls within I_1 at any one time-step *must* switch to I_0 at the next time-step. However, the converse is not true, resulting in the fact that the itineraries generated by this map cannot contain the block of symbols (11). A number expressed in this base could be converted back to decimal (at least to some desired accuracy since γ is a non-terminating irrational number in decimal) using the familiar formula for positional notation:

$$x = \gamma^n d_n + \gamma^{n-1} d_{n-1} + \cdots + \gamma^2 d_2 + \gamma^1 d_1 + d_0 + \gamma^{-1} d_{-1} + \gamma^{-2} d_{-2} + \cdots + \gamma^{-m} d_{-m}, \qquad (10)$$

where $d_i \in \mathbb{Z}_2$ since in beta expansions the alphabet is formed by the integers starting at zero that fit within the base, and $\gamma = 1.61...$. Thus, the above number would be indistinguishable from a binary number, based on the notation alone, which makes it necessary to indicate explicitly with a subscript when one is using a non-integer base: 10110001.1010_γ. The decimal point is immediately to the right of the d_0 digit, and negative powers indicate fractional digits.

Before we leave the discussion of possible connections between dynamical systems and number systems, since the logistic map is a smooth curve (a parabola), to develop the concept of a generalised number system we would need to look for the 'base' of the expansion of any one starting point $x_0 \in [0, 1]$. Using the usual calculus argument the smooth curve can be approximated as a series of linear segments, to each of which we would assign a corresponding interval $I_i,\ i = 0, ..., n$, and we would then let n go to infinity and the size of the intervals go to zero. This would lead to a rather strange number system with an infinity of bases (one of each value of the tangent to the parabola) and an infinite alphabet, something that may not even definable formally.

But perhaps this is not quite necessary. In fact, for the logistic map we have already divided the unit interval into two sub-intervals, and we have already defined a convenient itinerary function $S : \Lambda \to \Sigma$ that approximates the infinite sequence of x-values from Λ into an infinite string of binary digits. Williams sums it up nicely:

> We see here an exchange of spatial information for time series information mediated by dynamics: We can recover the complexity of the continuum I from our crude 2-element partition, provided that we observe the evolution of the system for all time. [97]

So one reason for this laborious detour into number systems is that it enables us to hold two different pictures in our mind at the same time and for the same physical entity:

- The time-evolution of a dynamical variable such as population level starting at value x_0 and crudely discretised as a time series of binary levels.
- The same sequence of binary digits as an expansion of the same initial value x_0 in an unknown and mysterious base, i.e. the same time series interpreted as *a single number*.

These two perspectives will now help us introduce two more concepts that form the bedrock of symbolic dynamics in a way that will hopefully look intuitively connected with dynamical systems. The two concepts are the shift map and the equivalence of the shift map to the logistic map. 'Equivalence' is meant in a very specific sense that will require some unpacking, as we now show.

2.1.3 The shift map and its dynamical properties

Although in this report we are not going to be able to discover the mysterious base – or if it even exists – the second perspective in the itemised list above is useful to us because it helps us understand the shift map. In fact, if we apply the function we used to generate the binary expansion of a decimal number, Eq. (7), to the same binary number we generated, we will notice that the effect will be to shift the decimal point to the right by one position or, equivalently, the whole string to the left by one position, and drop the first digit. Because we are operating on objects that are already in Σ, we are not talking about the doubling map (7) here, but of a new map which we call $\sigma : \Sigma \to \Sigma$ such that

$$\sigma(s_0 s_1 s_2 \dots) = (s_1 s_2 s_3 \dots) \tag{11}$$

It turns out that this simple and seemingly innocent map is quite powerful. Recalling that $f_\lambda : \Lambda \to \Lambda$, we wish to construct σ with the following properties ([47]: 347):

- σ is chaotic
- σ on Σ is conjugate to f_λ on Λ
- σ is completely understandable from a dynamical systems point of view.

Proposition 2.5 ([47]: 347) The shift map $\sigma : \Sigma \to \Sigma$ is continuous.

Proof. Let $s = (s_0 s_1 s_2 \dots) \in \Sigma$, and let $\epsilon > 0$. Choose n so that $1/2^n < \epsilon$. Let $\delta = 1/2^{n+1}$. Suppose that $d(s,t) < \delta$, where $t = (t_0 t_1 t_2 \dots)$. By Proposition 2.1 his means that $s_i = t_i$ for the first $n+2$ symbols, i.e. $i = 0, \dots, n+1$. Now,

$$\sigma(s) = (s_1 s_2 \dots s_n s_{n+1} s_{n+2} s_{n+3} \dots) \tag{12}$$
$$\sigma(t) = (s_1 s_2 \dots s_n s_{n+1} t_{n+2} t_{n+3} \dots), \tag{13}$$

Meaning that the two new sequences are equal for the first $n+1$ symbols. Therefore, $d(\sigma(s), \sigma(t)) \leq 1/2^n < \epsilon$. Hence, σ is continuous. □

We need σ to be continuous in order to be able to form a homeomorphism with λ, which is needed to prove conjugacy:

Proposition 2.6 ([47]: 347) The itinerary function $S : \Lambda \to \Sigma$ is a conjugacy between the logistic map f_λ and the shift map σ.

Proof. Having already stated that S is a homeomorphism in Theorem 2.2, we just need to show that $S \circ f_\lambda = \sigma \circ S$. Recall that $f_\lambda : \Lambda \to \Lambda$ and $\sigma : \Sigma \to \Sigma$ and let $x_0 \in \Lambda$ and $S(x_0) = (s_0 s_1 s_2 \dots)$. Then,

$$x_0 \in I_{s_0}, \quad f_\lambda(x_0) \in I_{s_1}, \quad f_\lambda^2(x_0) \in I_{s_2}, \quad \dots \tag{14}$$

But then, $S(f_\lambda(x_0)) = (s_1 s_2 s_3 \dots)$, meaning that

$$S(f_\lambda(x_0)) = \sigma(S(x_0)), \tag{15}$$

which is kind of trivial in hindsight. □

We have shown the following:

$$\begin{array}{ccc} \Lambda & \xrightarrow{S} & \Sigma \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \sigma} \\ \Lambda & \xrightarrow{S} & \Sigma \end{array} \tag{16}$$

and, in particular,

$$\begin{array}{ccc} x_0 & \xrightarrow{S} & S(x_0) \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \sigma} \\ x_1 & \xrightarrow{S} & S(x_1). \end{array} \tag{17}$$

Hirsch et al. [47] make the point that the shift map is much more nicely behaved than the logistic map. Hence it is worth pushing the argument a bit more in order to show that these two functions are actually equivalent. By 'elevating' the shift map to the level of a dynamical system of equivalent complexity to f_λ, we will gain the ability to draw conclusions about f_λ by working with σ. This of course is the original motivator for the development of the field of symbolic dynamics in the first place.

One area where it is easy to see that σ is easier to work with is in the classification of the periodic points of the map. In fact, any periodic point of length n will have the form, $(\overline{s_0, s_1, s_2, \ldots, s_{n-1}})$. For any value of n, therefore, there will be 2^n periodic points. On the other hand, since $x \in \mathbb{R}$ for the logistic map, it would be horrendously difficult to find, let alone completely classify, the corresponding periodic points for f_λ. The reason we know that they exist and that there is a 1-1 correspondence between the two sets of periodic points is because we have just proven in Proposition 2.6 that f_λ and σ are conjugate.

We now show that σ is chaotic. Following Definition 2.3 we can construct a point in Σ whose orbit is dense as follows:

$$s^* = (0\ 1 \qquad 00\ 01\ 10\ 11 \qquad 000\ 001\ 010\ 011\ 100\ 101\ 110\ 111 \qquad \ldots) \tag{18}$$

Thus, s^* contains all the strings of any length, arranged in order of increasing size. Now assume that we want to match an arbitrary sequence t in its first n places. All we have to do is iterate $\sigma(s^*)$ a sufficient number of times, say k, until we find the block in s^* of length n that matches the first n symbols of t. This will ensure that

$$d(\sigma^k(s^*), t) \leq \frac{1}{2^{n-1}}. \tag{19}$$

By increasing n and k we can bring a future iterate of s^* as close as we want to any given string t. Therefore, s^* is dense under σ in Σ, which means that σ is transitive. As Hirsch et al. [47] explain we can create any number of other points that are dense in Σ, simply by rearranging the blocks shown in Equation (18). To identify a point in Λ whose orbit under f_λ is dense is unfathomable. This, therefore, is what we mean when we say that the dynamics of σ are 'completely understandable'.

Finally, we need to address sensitive dependence on initial conditions. Let's choose the constant β of Definition 2.3 as 2, since this is the largest distance possible between two points in Σ (see Equation (5)). We need to show that the distance between two points that are initially arbitrarily close will, after a suitable number of iterations, reach this maximum distance. To

this end we define the symbol $\hat{s}$ as meaning 'not s', i.e. if $s = 0$, $\hat{s} = 1$, and vice versa. Now we pick the point s' such that

$$s = (s_0 s_1 s_2 \dots s_n s_{n+1} s_{n+2} \dots)$$
$$s' = (s_0 s_1 s_2 \dots s_n \hat{s}_{n+1} \hat{s}_{n+2} \dots).$$

We thus have, by construction, that

$$d(s, s') = \frac{1}{2^n}$$
$$d(\sigma^{n+1}(s), \sigma^{n+1}(s')) = 2.$$

With the above, we have in fact proven

> **Theorem 2.7** ([47]: 348) The shift map σ is chaotic on Σ, and so by the conjugacy proven by Proposition 2.6 the logistic map is chaotic on Λ when $\lambda > 4$.

Taking stock of how far we have got, we have shown that symbolic dynamics provides a powerful analytical tool to investigate the properties of dynamical systems. However, it does not yet appear to enable us to *build* a dynamical system according to a *specification*. This is a long-term goal that we will continue to keep in mind. For the time being we will continue to build formalisations that can act as a bridge between dynamical systems and computing systems, classifying and documenting important properties as we encounter them. In this spirit we now spend a few words on higher-dimensional dynamical systems before switching the perspective to coding.

2.1.4 Homoclinic phenomena and the p53-mdm2 system

Chapter 16 of [47] discusses how a three-dimensional system can be reduced to a 2D map, leading eventually to an idealisation known as the horseshoe map. This is of interest because we would like to understand what symbolic dynamics could tell us about higher-dimensional dynamical systems such as the p53-mdm2 system. The character of the results is mathematical and similar to the above, so not yet close enough to machines and automata for our purposes. However, one point deserves to be made and certainly motivates further study in this area: it turns out that the simplified form of the p53-mdm2 model discussed in D1.3 [15] (i.e. with no radiation and hence only 3 variables) has a phase space trajectory that appears to be a homoclinic orbit. Figure 12 shows the phase space plot of the same data shown in Figure 15 of [15].

2.2 Symbolic dynamics from the point of view of coding

In this section we shift gears, in the sense that we describe symbolic dynamics from a more abstract point of view, which is more difficult to relate directly to dynamical systems. However, the relationship with the results and the concepts we just finished discussing should be very clear. The material in this section follows closely the first few chapters of Lind and Marcus's textbook [62]. The objective of this section is to show how a particular type of shift space corresponds to finite-state automata.

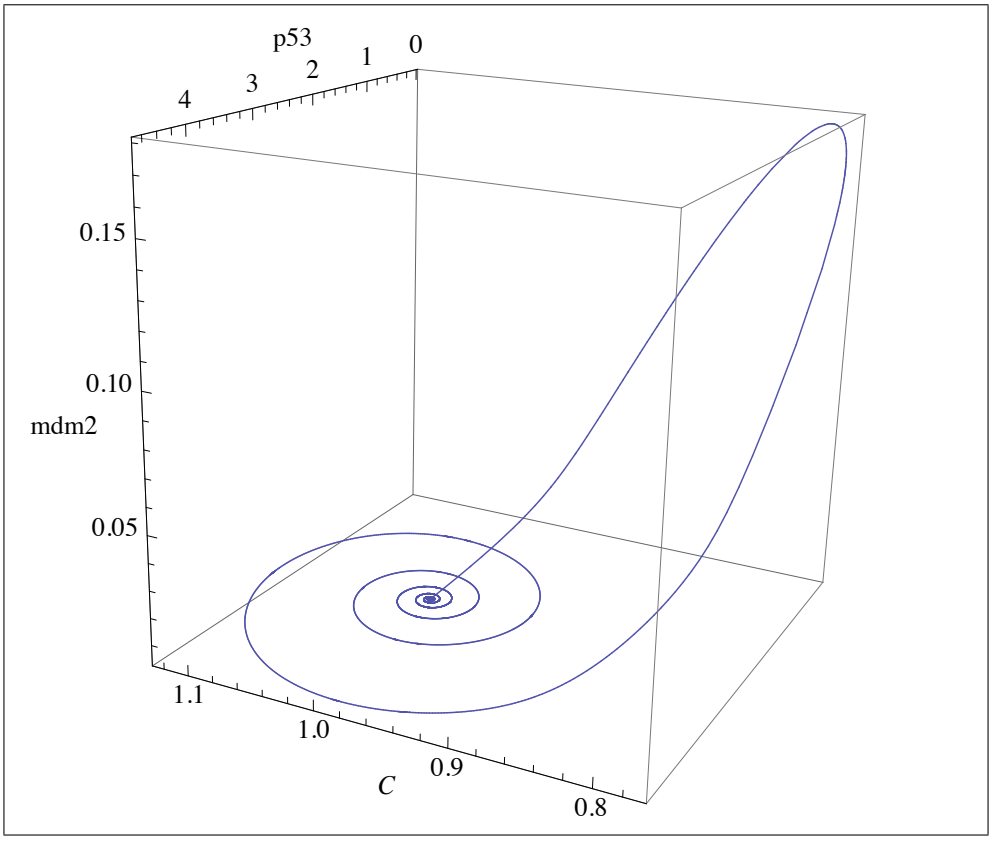


Figure 12: **Phase space trajectory of p53-mdm2 system exhibiting homoclinic behaviour**

2.2.1 Basic definitions

We begin by generalising symbolic sequences so that they are *bi*-infinite. Given an alphabet $\mathcal{A}$ we define a collection of bi-infinite sequences from $\mathcal{A}$

$$x = (x_i)_{i\in\mathbb{Z}}. \tag{20}$$

Each x_i is a coordinate of x, so x can be thought of as an infinite-dimensional vector. For convenience we use the decimal point to pick up the 0^{th} coordinate:

$$\ldots x_{-3}x_{-2}x_{-1}.x_0x_1x_2x_3\ldots \tag{21}$$

The collection of *all* bi-infinite sequences from $\mathcal{A}$ is called the **full $\mathcal{A}$-shift**:

$$\mathcal{A}^{\mathbb{Z}} = \{x = (x_i)_{i\in\mathbb{Z}} : x_i \in \mathcal{A}\ \forall i \in \mathbb{Z}\}. \tag{22}$$

We now list a few definitions that will be needed:

- Each $x \in \mathcal{A}$ is a *point* in the full shift.
- A **block** is a finite sequence of symbols from $\mathcal{A}$.
- The **length** of a block u is the number of symbols it contains, denoted by $|u|$.
- The empty block is ϵ.
- A k-block has length k.
- The set of all k-blocks over $\mathcal{A}$ is denoted $\mathcal{A}^k$.

- If $x \in \mathcal{A}^{\mathbb{Z}}$ and $i \leq j$ then $x_{[i,j]} = x_i x_{i+1} \ldots x_j$.
- If $i > j$ then $x_{[i,j]} = \epsilon$.
- $x_{[i,\infty)}$ is a right-infinite sequence (not a block).
- The central $(2k+1)$ block of x is $[-k, k]$.
- u^n denotes the concatenation of n copies of u.
- $u^0 = \epsilon$.
- $u^m u^n = u^{m+n}$.
- $u^\infty = \ldots uuuuu.uuuuuu \ldots$.
- If the decimal point is associated with the present, then the passage of time corresponds to shifting the sequence one place to the left.

The shift map σ on the full shift $\mathcal{A}^{\mathbb{Z}}$ maps a point x to the point $y = \sigma(x)$ such that $y_i = x_{i+1}$. Thus, $\sigma : \mathcal{A}^{\mathbb{Z}} \to \mathcal{A}^{\mathbb{Z}}$. We now list a few properties of σ, also useful for what follows:

- σ is both 1-1 and onto.
- $\sigma^k = \sigma \circ \sigma \circ \sigma \circ \ldots$ k times, so it shifts x k slots to the left.
- A code transforms a given sequence into another.
- **Stationary codes** do not change with time. For example, take $\phi : \{0,1\}^{\mathbb{Z}} \to \{0,1\}^{\mathbb{Z}}$,

$$\phi(x) = y : y_i = x_i + x_{i+1} \mod 2. \tag{23}$$

  This function acts the same regardless of the value of i, hence it is stationary.

- A stationary rule commutes with σ: $\sigma \circ \phi = \phi \circ \sigma$, or

$$\begin{array}{ccc} x & \xrightarrow{\sigma} & \sigma(x) \\ {\scriptstyle\phi}\downarrow & & \downarrow{\scriptstyle\phi} \\ \phi(x) & \xrightarrow{\sigma} & \underset{=\sigma(\phi(x))}{\phi(\sigma(x))} \end{array}. \tag{24}$$

- A point x is **periodic** for σ if $\exists (n > 1) : \sigma^n(x) = x$. Such a point must have the form u^∞ for some block u. The period of x is the length of u.
- If $\sigma(x) = x$, $\quad x$ is a **fixed point** of σ. Such a point must have the form a^∞ for some $a \in \mathcal{A}$.

2.2.2 Shift spaces

A **shift space** is a subset of points of a full shift, satisfying a fixed set of constraints. To be more precise, we follow Williams's definition of a shift space as a *pair* (X, σ_X), where X is a closed subset of a full shift, invariant under the action of σ for that full shift, and σ_X is the restriction of that map to X [97]. However, in common usage we tend to refer to X by itself as a shift space. We continue with the style of bulleted lists since it is quite efficient at communicating basic definitions.

- If $x \in \mathcal{A}$ and w is a block over $\mathcal{A}$, w occurs in x if $\exists i, j : x_{[i,j]} = w$
- We define $\mathcal{F}$ as a set of forbidden blocks.

- $X_{\mathcal{F}}$ is the subset of points in $\mathcal{A}^{\mathbb{Z}}$ that do not contain any blocks $\in \mathcal{F}$. Hence $X_{\mathcal{F}}$ is a shift space. We will use X and $X_{\mathcal{F}}$ interchangeably, against Lind and Marcus's advice, if the meaning is clear from the context.
- $\mathcal{F}$ may be finite or infinite, although we will work only with the former. The reason is that a finite set of forbidden blocks gives rise to so-called 'shifts of finite type'. A special kind of these shifts are the sofic shifts, which correspond to finite-state automata and regular languages. Although this may seem very restrictive, the point (also discussed in [81]) is that interacting automata can give us plenty of expressiveness, including models of cell metabolism. Therefore, it seems better to work with somewhat simpler objects since they appear to be sufficient and in any case figuring out how to get them to interact will be enough of a challenging problem in itself.
- For a given shift space in principle there could be more than one $\mathcal{F}$.
- If $\mathcal{F} = \mathcal{A}$ then the shift space is $\emptyset$.
- If $X = \mathcal{A}^{\mathbb{Z}}$ then $\mathcal{F} = \emptyset$.
- X contained in Y means that X is a subshift of Y.
- Example. If $\mathcal{A} = \{0, 1\}$ and $\mathcal{F} = \{11\}$, $X_{\mathcal{F}}$ is called the **golden mean shift**, for reasons that should be clear from Section 2.1.2.
- Example. For $\mathcal{A} = \{0, 1\}$, a **run-length limited shift**, denoted by $X(d, k)$, is a set of sequences in which 1s appear infinitely often in both directions, each surrounded by at least d 0s but no more than k 0s. In this case, $\mathcal{F} = \{0^n, (n < d) \cup (n > k); 1^m, m > 1\}$.
- Shift spaces are shift-invariant: $\sigma(X_{\mathcal{F}}) = X_{\mathcal{F}}$.
- σ_X is the restriction to X of σ acting on a full shift, which sounds more difficult than it is. If σ acts on X, it is σ_X automatically, it can't be anything else.
- Any subset of a full shift that is not shift-invariant is not a shift space.
- Shift invariance is necessary but not sufficient for a subset of a full shift to be a shift space.
- Some shift spaces have no periodic points.

2.2.3 Languages

If we specify which blocks are allowed, rather than forbidden, we get a language. Let X be a subset of a full shift, and let $\mathcal{B}_n(X)$ denote the set of all the n-blocks that occur in the points that $\in X$. The **language** of X is the collection

$$\mathcal{B}(X) = \bigcup_{n=0}^{\infty} \mathcal{B}_n(X) \tag{25}$$

Not every collection of blocks is a language of a shift space. If C is a collection of blocks over $\mathcal{A}$, the **complement** of C relative to the set of all possible blocks over $\mathcal{A}$ is C^c. In the following proof, it is helpful to keep in mind that a language is a collection of blocks of symbols, whereas a shift space is a set of infinite symbol sequences.

Proposition 2.8 ([62]: 10)

1. Let X be a shift space and $\mathcal{L} = \mathcal{B}(X)$ be its language. If $w \in \mathcal{L}$ then
   a) every sub-block $s \subset w \in \mathcal{L}$
   b) $\exists u, v \in \mathcal{L} : uwv \in \mathcal{L}$

2. The languages of shift spaces are characterised by 1. In other words, if $\mathcal{L}$ is a collection of blocks over $\mathcal{A}$, then $\mathcal{L} = \mathcal{B}(X)$ for some X iff $\mathcal{L}$ satisfies 1.
3. The language of a shift space determines the shift space: $\forall X, X = X_{\mathcal{B}(X)^c}$, which means that a shift space is defined by a set of forbidden blocks equal to the complement of its language.

Proof.

1. If $w \in \mathcal{L} = \mathcal{B}(X)$, then $w \subset x \in X$. But then any $s \subset w \in x$. Thus $s \subset \mathcal{L}$ too. Further, since x has infinite length, $\exists u, v \in x : uwv \in X$. Therefore, $u, v \in \mathcal{L}$ and $uwv \in \mathcal{L}$.
2. Reverse implication: 'If $\mathcal{L}$ satisfies 1, then $\mathcal{L} = \mathcal{B}(X)$'. Let $\mathcal{L}$ be a collection of blocks satisfying 1, and $X = X_{\mathcal{L}^c}$. If $w \in \mathcal{B}(X)$, then $w \in X_{\mathcal{L}^c}$, which implies that $w \notin \mathcal{L}^c$, i.e. $w \in \mathcal{L}$. Therefore, $\mathcal{B}(X) \subseteq \mathcal{L}$.
   Forward implication: 'If $\mathcal{L} = \mathcal{B}(X)$, then $\mathcal{L}$ satisfies 1'. Now suppose that $w = x_0 x_1 \dots x_m \in \mathcal{L}$. Then by 1b $uwv \in \mathcal{L}$, where $u = x_{-p} x_{-p+1} \dots x_{-1}$ and $v = x_{m+1} x_{m+2} \dots x_{m+q}$. Letting $p \to -\infty$ and $v \to \infty$, we can show that every sub-block of $x \in \mathcal{L}$. This is possible because by saying that $w \in \mathcal{L}$ we imply that $w \in x \in X$, i.e. the conditions of 1 apply. If every sub-block of $x \in \mathcal{L}$, then $x \in X_{\mathcal{L}^c}$. Since $w \in x$, then $w \in \mathcal{B}(X_{\mathcal{L}^c}) = \mathcal{B}(X)$. Thus, $\mathcal{L} \subseteq \mathcal{B}(X)$. Hence, $\mathcal{L} = \mathcal{B}(X)$.
3. To prove 3 we show, in essence, that the set of forbidden blocks of a shift space is equal to the complement of its language. We do this in two steps:
   If $x \in X$, $x \in X_{\mathcal{B}(X)^c}$: hence $X \subseteq X_{\mathcal{B}(X)^c}$

   If $x \in X$, no $w \subset x$ can $\in \mathcal{B}(X)^c$. Hence, $x \in X_{\mathcal{B}(X)^c}$.

   If $x \in X_{\mathcal{B}(X)^c}$, $x \in X_{\mathcal{F}}$: hence $X_{\mathcal{B}(X)^c} \subseteq X_{\mathcal{F}} = X$

   Since X is a shift space, $\exists \mathcal{F} : X = X_{\mathcal{F}}$. If $x \in X_{\mathcal{B}(X)^c}$ then every $w \subset x$ must $\in \mathcal{B}(X) = \mathcal{B}(X_{\mathcal{F}})$, and thus $w \notin \mathcal{F}$. Hence, $x \in X_{\mathcal{F}}$.

   Therefore, $X = X_{\mathcal{B}(X)^c}$.

□

The third part of this proposition has shown that there is a largest set of forbidden blocks that defines a shift space. In other words, even though there could be different sets of forbidden blocks defining the *same* shift space, there is a largest collection $\mathcal{B}(X)^c$, the complement of the language of X, that is the largest collection of forbidden blocks possible. Therefore, it uniquely defines the language it corresponds to. The 1-1 correspondence between languages $\mathcal{L}$ and shift spaces X that satisfy point 1 is summarised by the equations:

$$\mathcal{L} = \mathcal{B}(X_{\mathcal{L}^c}) \qquad X = X_{\mathcal{B}(X)^c} \tag{26}$$

Another consequence of part 3 of the proof above is that it enables the characterisation of a shift space in terms of allowed blocks.

Corollary 2.8.1 ([62]: 10) Let X be a subset of the full $\mathcal{A}$-shift. Then X is a shift space iff whenever $x \in \mathcal{A}^{\mathbb{Z}}$ and each $x_{[i,j]} \in \mathcal{B}(X)$ then $x \in X$.

2.2.4 Higher block shifts

We can define a new alphabet where each letter is a block from the original shift. Let X be a shift space over alphabet $\mathcal{A}$ and $\mathcal{A}_X^{[N]} = \mathcal{B}_N(X)$ be a collection of allowed N-blocks in X. We consider $\mathcal{A}_X^{[N]}$ as an alphabet in its own right, with full shift $\left(\mathcal{A}_X^{[N]}\right)^{\mathbb{Z}}$. The N^{th} higher block code $\beta_N : X \to \left(\mathcal{A}_X^{[N]}\right)^{\mathbb{Z}}$ is

$$(\beta_N(x))_i = x_{[i,i+N-1]}. \tag{27}$$

Thus, β_N replaces the i^{th} coordinate of x with the block of length N that starts at i. Lind and Marcus suggest writing the blocks vertically. For example, for $N = 4$, we have

$$\beta_4(x) = \ldots \begin{bmatrix} x_0 \\ x_{-1} \\ x_{-2} \\ x_{-3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_0 \\ x_{-1} \\ x_{-2} \end{bmatrix} \begin{bmatrix} x_2 \\ x_1 \\ x_0 \\ x_{-1} \end{bmatrix} . \begin{bmatrix} x_3 \\ x_2 \\ x_1 \\ x_0 \end{bmatrix} \begin{bmatrix} x_4 \\ x_3 \\ x_2 \\ x_1 \end{bmatrix} \ldots \qquad \in \left(\mathcal{A}_X^{[4]}\right)^{\mathbb{Z}} \tag{28}$$

Let X be a shift space. The N^{th} **higher block shift** $X^{[N]}$, or **higher block presentation** of X, is the image $X^{[N]} = \beta_N(X)$ in the full shift over $\mathcal{A}_X^{[N]}$. If u and v are N-blocks, $u = u_1 u_2 \ldots u_N$ and $v = v_1 v_2 \ldots v_N$, they **overlap progressively** if $u_i = v_{i-1}$, for $1 < i \leq N$:

$$u_1 \boxed{\begin{array}{cccccc} u_2 & u_3 & u_4 & \ldots & u_{N-1} & u_N \\ v_1 & v_2 & v_3 & \ldots & v_{N-2} & v_{N-1} \end{array}} v_N \tag{29}$$

uv is a 2-block over $\mathcal{A}_X^{[N]}$. If it is present in the image point $\beta_N(x)$ of x, then u and v must overlap progressively. To keep the new symbols straight let's add some clarifications on the notation:

- $\mathcal{A}^{\mathbb{Z}}$: all possible sequences over $\mathcal{A}$ (full shift).
- $\mathcal{A}^N = \mathcal{A}^{[N]}$: all possible blocks of length N over $\mathcal{A}$. Note: $\left|\mathcal{A}^{[N]}\right| = |\mathcal{A}|^N$.
- $\mathcal{A}_X^{[N]}$: all blocks of length N that are allowed in X, thereby forming a new alphabet.
- $\left(\mathcal{A}_X^{[N]}\right)^{\mathbb{Z}}$: full shift over new alphabet $\mathcal{A}_X^{[N]}$.
- $X^{[N]}$: N^{th} higher block shift is the overlapping-block representation of $\left(\mathcal{A}_X^{[N]}\right)^{\mathbb{Z}}$; also written as $\beta_N(X)$, where $\beta_N : X \to \left(\mathcal{A}_X^{[N]}\right)^{\mathbb{Z}}$.
- x: a point $\in X$
- $\beta_N(x)$: a point $\in \left(\mathcal{A}_X^{[N]}\right)^{\mathbb{Z}}$

We might summarise the relationships between some of these different sets as follows:

$$\underbrace{X^{[N]} \quad \subseteq \quad \left(\mathcal{A}_X^{[N]}\right)^{\mathbb{Z}}}_{\substack{\text{The additional constraint} \\ \text{is the requirement of} \\ \text{progressive overlap}}} = \underbrace{X \quad \subseteq \quad \mathcal{A}^{\mathbb{Z}} = \left(\mathcal{A}^{[N]}\right)^{\mathbb{Z}}}_{\substack{\text{The constraint here} \\ \text{is just the set of} \\ \text{forbidden blocks } \mathcal{F}}}, \tag{30}$$

where the equal signs should not be interpreted literally. They indicate the fact that the different 'presentations' are equivalent in the sense that, for example, x can be reconstructed from $\beta_N(x)$ by picking the bottom element in each column shown in Equation 28. The subset relationships, on the other hand, are the consequence of imposing additional constraints as one moves right to left in the equation.

Proposition 2.9 ([62]: 13) The higher block shifts of a shift space are also shift spaces.

Proof. Let X be a shift space over $\mathcal{A}$ and $N \geq 1$. Then, $\exists \mathcal{F} : X = X_{\mathcal{F}}$. Choose N and create a new $\widetilde{\mathcal{F}}$ so that *each* $u \in \mathcal{F}$ is replaced by *all* N-blocks over $\mathcal{A}$ containing u. So the blocks of $\widetilde{\mathcal{F}}$ will all be of length N and will contain all the blocks of $\mathcal{F}$. Now let $w = (a_1 a_2 a_3 \dots a_m) \in \widetilde{\mathcal{F}}$, where $m \geq N$ necessarily. For each w, let

$$w^{[N]} = \begin{bmatrix} a_N \\ \vdots \\ a_2 \\ a_1 \end{bmatrix} \begin{bmatrix} a_{N+1} \\ \vdots \\ a_3 \\ a_2 \end{bmatrix} \dots \begin{bmatrix} a_m \\ \vdots \\ a_{m-N+2} \\ a_{m-N+1} \end{bmatrix} \tag{31}$$

be the corresponding block over $\mathcal{A}^N$, of length $m - N + 1$. Let $\mathcal{F}_1$ be the set of all blocks $w^{[N]}$ over $\mathcal{A}^N$, for some $w \in \widetilde{\mathcal{F}}$. This means that we can pick any $w \in \widetilde{\mathcal{F}}$ and for each build $w^{[N]}$. So $\mathcal{F}_1$ is the set of N-blocks that corresponds to $\widetilde{\mathcal{F}}$, and therefore $\mathcal{F}$. But, as we saw, we also have other constraints. Thus, $X^{[N]} \subseteq X_{\mathcal{F}_1}$. In fact, the points in $X^{[N]}$ also need to satisfy the progressive overlap condition. Thus,

$$\mathcal{F}_2 = \{uv : u, v \in \mathcal{A}^N, \text{ and } u \text{ and } v \text{ do not overlap progressively}\}. \tag{32}$$

Clearly, $X^{[N]} \subseteq X_{\mathcal{F}_2}$. Thus, $X^{[N]} \subseteq X_{\mathcal{F}_1} \cap X_{\mathcal{F}_2}$. We assume without proof that $X_{\mathcal{F}_1} \cap X_{\mathcal{F}_2} = X_{\mathcal{F}_1 \cup \mathcal{F}_2}$. Then, $\underline{X^{[N]} \subseteq X_{\mathcal{F}_1 \cup \mathcal{F}_2}}$.

Summarising what we have said so far in this proof, $X^{[N]}$ is built as the image of X using map $\beta_N(X)$ and is a subset of the full shift over $\mathcal{A}_X^{[N]}$, $\left(\mathcal{A}^{[N]}\right)^{\mathbb{Z}}$. $X_{\mathcal{F}_1 \cup \mathcal{F}_2}$, on the other hand, is a shift space obtained from the original X by forbidding

- the same blocks in $\mathcal{F}$
- the blocks that do not overlap progressively.

So $X_{\mathcal{F}_1 \cup \mathcal{F}_2}$ is a smaller set than $X_{\mathcal{F}}$. It is built like $X^{[N]}$, but it is actually a shift space over the original $\mathcal{A}$. Hence we can be sure it is a shift space. The proof strategy, then, is to show that $X^{[N]} = X_{\mathcal{F}_1 \cup \mathcal{F}_2}$, thus establishing that $X^{[N]}$ is a shift space too. We have so far shown that $X^{[N]} \subseteq X_{\mathcal{F}_1 \cup \mathcal{F}_2}$. If we can show the converse, we are done.

Suppose $y \in X_{\mathcal{F}_1 \cup \mathcal{F}_2}$, so y is one of these sequences of blocks arranged vertically. Let x be the point of $\mathcal{A}^{\mathbb{Z}}$ that we can reconstruct by taking all the symbols at the bottom of the vertical stacks. Then $x \in X = X_{\mathcal{F}}$ since y by assumption satisfies the constraints from $\mathcal{F}_1$; further, $y = \beta_N(x)$ since it also satisfies the constraints from $\mathcal{F}_2$. As we say above, $X_{\mathcal{F}_1 \cup \mathcal{F}_2}$ is drawn from $\mathcal{A}$ not $\mathcal{A}_X^{[N]}$. So we can pick a whole series of points y that definitely belong to a shift space (smaller than $X_{\mathcal{F}}$), and in each case we can show that these points satisfy the constraints of $X^{[N]}$. Therefore, $\underline{X_{\mathcal{F}_1 \cup \mathcal{F}_2} \subseteq X^{[N]}}$.

Hence, $X^{[N]} = X_{\mathcal{F}_1 \cup \mathcal{F}_2}$. □

2.2.5 Sliding block codes

Let $x = (\dots x_{-2} x_{-1} x_0 x_1 x_2 \dots)$ be a sequence of symbols in a shift space over $\mathcal{A}$. We can transform x into a new sequence $y = (\dots y_{-2} y_{-1} y_0 y_1 y_2 \dots)$ over a different alphabet $\mathcal{U}$ as follows. Pick integers m and n, with $-m \leq n$. Let y_i depend on a function Φ that depends on the window of x-coordinates from $i - m$ to $i + n$.[9] We call

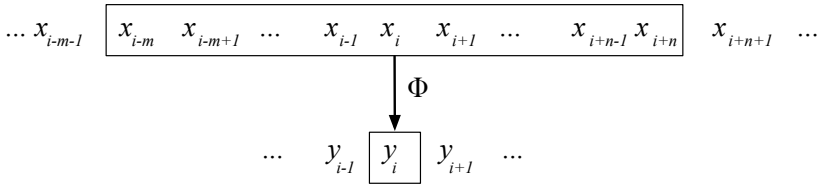
$$\Phi : \mathcal{B}_{m+n+1}(X) \to (U) \tag{33}$$

a **block map** from allowed $(m + n + 1)$-blocks in X to symbols in $\mathcal{U}$:

$$y_i = \Phi(x_{[i-m, i+n]}) \tag{34}$$

[9] We are following Lind and Marcus, but this is slightly sloppy. If m is a negative number then it should be from $i + m$ to $i + n$. Treating m as a positive number won't cause any problems in what follows.

Definition 2.10 Let X be a shift space over $\mathcal{A}$ and $\Phi : \mathcal{B}_{m+n+1}(X) \to (U)$ be a block map. Then $\phi : X \to (U)^{\mathbb{Z}}$ defined by $y = \phi(x)$ with y_i as above is called the **sliding block code** with memory m and anticipation n induced by Φ. The formation of ϕ from Φ is denoted by $\phi = \Phi_{\infty}^{[-m,n]}$, or Φ_{∞} if we know m and n already. If m is not specified, $m = 0$. If Y is a shift space in the full shift $\mathcal{U}^{\mathbb{Z}}$ and $\phi(X) \subseteq Y$, then $\phi : X \to Y$.



- If $m = n = 0$ then we have a 1-block code $x_i \to y_i$.
- Sliding block codes commute with the shift map:

$$\begin{array}{ccc} X & \xrightarrow{\sigma_X} & X \\ {\scriptstyle\phi}\downarrow & & \downarrow{\scriptstyle\phi} \\ Y & \xrightarrow{\sigma_Y} & Y. \end{array} \tag{35}$$

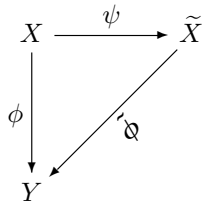
- **Proposition 2.11** Let X and Y be shift spaces. A map $\phi : X \to Y$ is a sliding block code iff $\phi \circ \sigma_X = \sigma_Y \circ \phi$ and $\exists N \geq 0 : \phi(x)_0$ is a function of $x_{[-N,N]}$. In other words, $\phi(x)_0$ depends only on a central block of x.
- If $\phi : X \to Y$ is onto, it is called a **factor code**, and Y is a factor of X.
- If $\phi : X \to Y$ is 1-1, it is called an **embedding** of X into Y.
- If $\phi : X \to Y$ has an inverse, it is unique.
- **Definition 2.12** A sliding block code $\phi : X \to Y$ is a **conjugacy** from X to Y if it is invertible. $X \cong Y$ means that X and Y are conjugate.
- If X and Y are conjugate they are actually the same object. Thus conjugacy is analogous to isomorphism. For example, X and $X^{[N]}$ are conjugate because we can find an inverse map $\beta_N^{-1} = \psi = \Psi_{\infty} : X^{[N]} \to X$, where $\Psi : \mathcal{A}_X^{[N]} \to \mathcal{A}$ is given by the function $\Psi(a_0 a_1 a_2 \dots a_{N-1}) = a_0$. To say that these two spaces are conjugate may appear to contradict Equation 30, since there we went out of our way to explain why $X^{[N]} \subseteq X$. But this just shows that conjugacy does not imply equality.
- **Proposition 2.13** Let $\phi : X \to Y$ be a sliding block code. If $x \in X$ has period n under σ_X, then $\phi(x)$ has period n under σ_Y, and the least period of $\phi(x)$ divides the least period of X. Embeddings, and therefore also conjugacies, preserve the least period of a point.

  Proof. $\sigma_X^n(x) = x. \qquad \sigma_Y^n(\phi(x)) = \phi(\sigma_X^n(x)) = \phi(x)$ □

- This proposition implies that, for each n, the number of points of period n is the same for all the shifts conjugate to a given shift. Thus, it is an invariant of the shift.
- Let $\phi : X \to Y$ be a sliding block code. We can recode X to a conjugate shift $\widetilde{X}$ such that $\tilde{\phi} : \widetilde{X} \to Y$ is a 1-block code. This is proven in the next proposition.

Proposition 2.14 ([62]: 19) Let $\phi : X \to Y$ be a sliding block code. Then there exists a higher block shift $\widetilde{X}$ of X, a conjugacy $\psi : X \to \widetilde{X}$, and a 1-block code $\tilde{\phi} : \widetilde{X} \to Y$ so that

$\tilde{\phi} \circ \psi = \phi$.



Proof. Suppose ϕ is induced by a block map Φ with memory m and anticipation n. Let $\mathcal{U} = \mathcal{B}_{m+n+1}(X)$ and define $\Psi : X \to \mathcal{U}^{\mathbb{Z}}$ by $\psi(x)_i = x_{[i-m,i+n]}$. The significance of this definition is that the blocks of the higher block shift have the same size as the block map of the sliding block code. Then, $\psi = \sigma_Y^{-m} \circ \beta_{m+n+1}$, i.e. we build the higher block code and then we shift it to the right by m, until x_{-m} is next to the decimal point.[10] The action of σ on Y shifts whole blocks, since the symbols of Y are blocks of length $m+n+1$, but the overlapping condition has as a consequence that the effect is the same as if σ were acting on the original symbol set, of X.

$$\ldots \begin{bmatrix} x_{-m-1} \\ \vdots \\ \end{bmatrix} . \begin{bmatrix} x_n \\ \vdots \\ x_{-m+1} \\ x_{-m} \end{bmatrix} \begin{bmatrix} x_{n+1+m+n+1} \\ \vdots \\ x_{n+1+1} \\ x_{n+1} \end{bmatrix} \ldots \tag{36}$$

So we have built $\widetilde{X} = \psi(X) = X^{[m+n+1]}$, which is a shift space by Proposition 2.9. Since σ and β are conjugacies, so is ψ. Now we can obtain $\tilde{\phi}$ simply as $\tilde{\phi} = \phi \circ \psi^{-1}$. What we have done is to build a higher block shift $\widetilde{X}$ conjugate to X such that its elements have the same size as the block map of ϕ. Therefore, $\tilde{\phi}$ is now a 1-block code. □

Theorem 2.15 ([62]: 20) The image of a shift space under a sliding block code is a shift space.

Proof. Let X and Y be shift spaces and $\phi : X \to Y$ a sliding block code. Using the previous proposition, we can assume that ϕ is a 1-block code. Let Φ be a 1-block map inducing ϕ. Now we perform a sleight of hand. We let the language of X be $\mathcal{L} = \{\Phi(w) : w \in \mathcal{B}(X)\}$, which needs to be explained with some care. First, Φ is supposed to be a 1-block map. Therefore, $\Phi(w)$, where w is a block of symbols of length ≥ 1, must imply repeated application of Φ in order to achieve the full mapping of w to $\Phi(w)$. The second point is trickier. The language of a shift space is defined as the union of all the blocks that appear within its points, i.e. $\mathcal{B}(X)$, *not* as the set of blocks that result from applying some arbitrary function to $\mathcal{B}(X)$. By applying an arbitrary function we are creating another set of blocks $\Phi(\mathcal{B}(X))$. Lind and Marcus are telling us that we can regard these new blocks as tantamount to the language of X as originally defined. Although they do not mention any concerns about whether Φ is bijective or invertible, there may not be any problem to work with this, in principle, except for the fact that ϕ happens to be the map from X to Y. Thus, the definition of $\mathcal{L}$ above is equivalent to saying that the language of X is made up of the blocks found in Y. It is reasonable to expect that the union of these blocks would then *also* represent the language of Y. If this is the case by assumption then it's small wonder that we are able to prove that Y is a shift space: it almost seems like we are relying on a tautology for our proof. But let's see how the proof develops. We need to show that $\phi(X) = X_{\mathcal{L}^c}$, which means that Y is defined by a set of forbidden blocks derived from the complement of its language. As we saw in Proposition 2.8 this is enough to show that $Y = \phi(X)$ is a shift space.

If $x \in X$, then every block in $\phi(X) \in \mathcal{L}$, so that $\phi(X) \in X_{\mathcal{L}^c}$. This proves that $\underline{\phi(X) \subseteq X_{\mathcal{L}^c}}$.

The converse is rather more complicated. We begin by letting $y \in X_{\mathcal{L}^c}$. We need to show that any such y is equal to $\phi(x)$, for some $x \in X$, as this would imply that $X_{\mathcal{L}^c} \subseteq \phi(X)$, which is the other part we need in order to show equality between them. Lind and Marcus

[10] The subscript Y on σ does not appear in [62] but Prof Lind suggested that adding it would make the explanation clearer [private communication].

show this by a method that they describe as analogous to Cantor's diagonal argument for the existence of uncountable sets. If $y \in X_{\mathcal{L}^c}$, then we know that the central $(2n+1)$-block of y is the image under Φ of the central $(2n+1)$-block of some point x, that we label $x^{(n)}$, because this is our starting assumption in how we defined $\mathcal{L}$. Thus,

$$\Phi\left(x^{(n)}_{[-n,n]}\right) = \phi\left(x^{(n)}\right)_{[-n,n]} = y_{[-n,n]}. \tag{37}$$

Now we need to extend n to ∞ and show that ultimately $y = \phi(x)$. To this end, consider the possible values of the 0^{th} coordinate of $x^{(n)}$, i.e. $x_0^{(n)}$, for $n \geq 1$. Since there are finitely many symbols in $\mathcal{A}$ (e.g. 2), there is an infinite set S_0 of integers for which $x_0^{(n)}$ is the same $\forall n \in S_0$. What this means is that we have only a finite choice of symbols for $x_0^{(n)}$, say 2, and therefore it will be 'easy' to find an infinite number of sequences, each with a greater value for n, such that $x_0^{(n)}$ is always the same, for example 0. If, by contrast, $\mathcal{A}$ were infinite, then $x_0^{(n)}$ would be able to assume any one of an infinite number of values. In that case it would be much harder to find an infinite number of sequences for which $x_0^{(n)}$ was the same symbol. Not sure if this statement is based on probability or some kind of measure theory argument, but it makes sense intuitively.

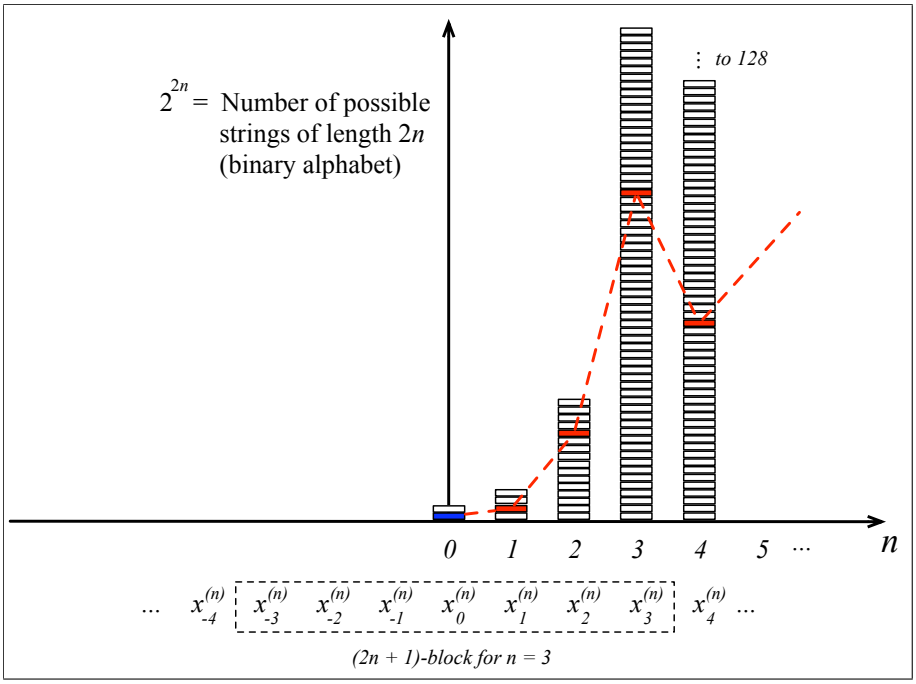


Figure 13: **Visualisation of Cantor's diagonal argument for Theorem 2.15,** $k = 0$

It is interesting to note that to each integer in the set S_0 there corresponds *one* possible central $(2n+1)$-block $\in x^{(n)}$, although 2^{2n} are possible.[11] Here then we start to see why Cantor's diagonal argument might be relevant. Figure 13 shows a possible visualisation of the process of populating S_0 as n grows. For each value of n the 2^{2n} possible blocks, whose $x_0^{(n)}$ value remains the same (shown as the blue box at the origin), are depicted as a stack of thin rectangles. But we only need to pick one for each n, as n grows to infinity, thereby traversing a 'diagonal' through the growing array of possible blocks.

The next step is to repeat this process but fixing one of the 8 possible central 3-blocks, as shown in Figure 14. The integers indexing the sizes of the central blocks which contain the same 3-block form an infinite subset of S_0, which we call $S_1 \subseteq S_0$. We keep doing this to

[11] For example, for $n = 3$ the central block has length 7, so it might appear that 128 blocks are possible. But $x_0^{(n)}$ is constrained to remain fixed, so there are only 6 bits available. Although the $x_0^{(n)}$ slot might seem to split the string into 2 3-bit strings, there are still 6 bits available, so 2^6 possible blocks.

an arbitrary size $(2k+1)$ of the 'frozen' central block $x^{(n)}_{[-k,k]}$, to which there corresponds an infinite subset $S_k \subseteq S_{k-1} \subseteq S_{k-2} \cdots \subseteq S_1 \subseteq S_0$.

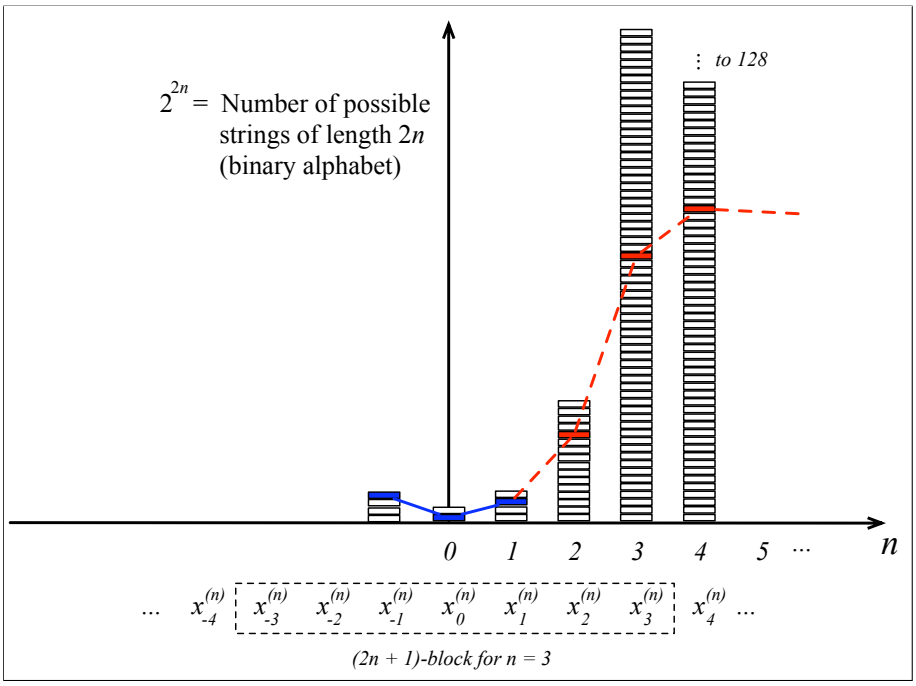


Figure 14: **Visualisation for Theorem 2.15,** $k = 1$

Now we define x to be the sequence with $x_{[-k,k]} = x^{(n)}_{[-k,k]}, \forall n \in S_k$. This means, as before, that for any k we take all the blocks defined by $n \geq k$ that give the same $x^{(n)}_{[-k,k]}$ but, this time, as we increment k we want $x^{(n)}_{[-k,k]}$ to match the *same* x. In other words, the blue patterns of frozen blocks do not change arbitrarily as we increment k, as they might have done before: now $x^{(n)}_{[-k,k]}$ remains the same within the larger $x^{(n)}_{[-k-1,k+1]}$. Having come this far now we notice that every block that could possibly appear within x can be contained within some interval $x_{[-k,k]}$, which we have just set equal to $x^{(n)}_{[-k,k]}$, which we know $\in \mathcal{B}(X)$ by construction. As a consequence, this particular x that we have built in this way must $\in X$, since X is a shift space and $\mathcal{B}(X)$ is its language. Finally, for each $k \geq 0$, $n \in S_k$, and $n \geq k$, we have

$$\Phi\left(x_{[-k,k]}\right) = \Phi\left(x^{(n)}_{[-k,k]}\right) = \phi\left(x^{(n)}\right)_{[-k,k]} = y_{[-k,k]}. \tag{38}$$

Since k is arbitrarily large, this shows that $\phi(x) = y$. Therefore, $X_{\mathcal{L}^c} \subseteq \phi(X)$. Hence, $X_{\mathcal{L}^c} = \phi(X)$ and we are done. (This proof has not been easy and I am not entirely sure I fully understand it.) □

Now let's assume we have a sliding block code $\phi : X \to Y$ that is 1-1, i.e. it is an embedding. By the theorem we just proved we know that $Y = \phi(X)$ is a shift space. Let $\psi : \phi(X) \to X$ be the inverse map, so that $x = \psi(y)$ whenever $y = \phi(x)$. The next theorem shows that ψ is a sliding block code.

Theorem 2.16 A sliding block code that is 1-1 and onto has sliding block inverse code, and is therefore a conjugacy.

Proof. We refer the reader to ([62]: 21). □

We originally meant to cover convolutional codes, since they fall within the framework of sliding block codes, but lack of time is preventing us from putting our notes to LaTeX. We must press on with the next topic. We only have time to touch on shifts of finite type and then sofic shifts.

2.2.6 Shifts of finite type

> Shift spaces described by a finite set of forbidden blocks are called shifts of finite type. Despite being the 'simplest' shifts, they play an essential role in mathematical subjects like dynamical systems. Their study has also provided solutions to important practical problems, such as finding efficient coding schemes to store data on computer disks. One reason why finite type shifts are so useful is that they have a simple representation using a finite, directed graph. Questions about the shift can often be phrased as questions about the graph's adjacency matrix. Basic results from linear algebra help us take this matrix apart and find the answers. ([62]: 28)

> **Definition 2.17** A **shift of finite type** is a shift space that can be described by a finite number of forbidden blocks.

If X is not a full shift, then it can also be defined by the complement of its language, $X = X_{\mathcal{B}(X)^c}$. Since the complement of a language is infinite, this shows that a shift of finite type can also be described by an *infinite* number of forbidden blocks. Definition 2.17 only says that there is *some* finite $\mathcal{F}$ that is sufficient.

Since $X_{\mathcal{F}}$ denotes the set of points $x \in \mathcal{A}^{\mathbb{Z}}$ that do not contain any blocks $w \in \mathcal{F}$, as we did in Proposition 2.9 we can easily define $\mathcal{F}_N$ as a set of blocks of length N, where N is the length of the longest block in $\mathcal{F}$, and still say that $X_{\mathcal{F}} = X_{\mathcal{F}_N}$. If all the blocks have length N then any $x \in \mathcal{A}^{\mathbb{Z}}$ will belong to $X_{\mathcal{F}}$ when $x_{[i,i+N-1]} \notin \mathcal{F}$, $\forall i \in \mathbb{Z}$; or, equivalently, when $x_{[i,i+N-1]} \in \mathcal{B}_N(X_{\mathcal{F}})$. Here $\mathcal{B}_N(X_{\mathcal{F}})$ is the set of all N-blocks that occur in $X_{\mathcal{F}}$. We don't need to worry about smaller blocks because they are already contained in the allowed N-blocks. For any block $> N$, the fractional part will fit in the next N-block. Hence knowing $\mathcal{B}_N(X_{\mathcal{F}})$ is sufficient.

> **Example 2.18: Even shift.** This shift space is the set of binary sequences such that between any two 1s there is an even number of 0s. Thus, for the even shift, $\mathcal{F} = \{10^{2n+1}1 : n \geq 0\}$. This shift is not of finite type since $\mathcal{F}$ is not finite. If it were, there would be an $N > 0$ and a finite collection $\mathcal{F}$ of N-blocks such that $X = X_{\mathcal{F}}$. Having set that up, now let $x = 0^{\infty}10^{2N+1}10^{\infty}$ (notice the capital N). Clearly, every N-block of x is in $\mathcal{B}_N(X)$, leading to $x \in X_{\mathcal{F}} = X$, contradicting the definition.

Shifts of finite type also have a memory.

> **Definition 2.19** A shift of finite type is **M-step** (or has memory M) if it can be described by a collection of forbidden blocks all of which have length $M + 1$.

Since a single $(M + 1)$-block $\in \mathcal{F}$ appearing in a point x makes x ineligible, any M-step shift is also a K-step shift $\forall K \geq M$. A 0-step shift is just a full shift, but with a smaller alphabet than it started with. 1-step shifts of finite type can be seen as a collection of symbols together with a description of which symbols can follow them, like a Markov chain without probabilities.

> **Proposition 2.20** If X is a shift of finite type, then $\exists M \geq 0 : X$ is M-step.
>
> *Proof.* Since X is of finite type, $X = X_{\mathcal{F}}$ for some finite $\mathcal{F}$. If $\mathcal{F}$ is not empty, let M be 1 less than the longest block of $\mathcal{F}$. We have already shown that $X = X_{\mathcal{F}_{M+1}}$, thus it is M-step. □

The next theorem sets the constraint to enable the splicing and joining of different segments, even belonging to different points of the same shift of finite type. The main concept is that for shifts of finite type words cannot necessarily just be joined end-to-end, they need to overlap a minimum amount. This theorem shows that this amount needs to be at least M.

Theorem 2.21 ([62]: 30) A shift space X is an M-step shift of finite type iff, whenever $uv, vw \in \mathcal{B}(X)$ and $|v| \geq M$, then $uvw \in \mathcal{B}(X)$.

Proof. If X is M-step, then whenever $uv, vw \in \mathcal{B}(X)$ and $|v| \geq M$ $uvw \in \mathcal{B}(X)$. $\mathcal{F}$ is made of $(M+1)$-blocks and is finite. Let $uv, vw \in \mathcal{B}(X)$, with $|v| = n \geq M$. Then there are two points $x, y \in X : x_{[-k,n]} = uv$, $y_{[1,l]} = vw$ and $x_{[1,n]} = y_{[1,n]} = v$. This is shown in Figure 15.

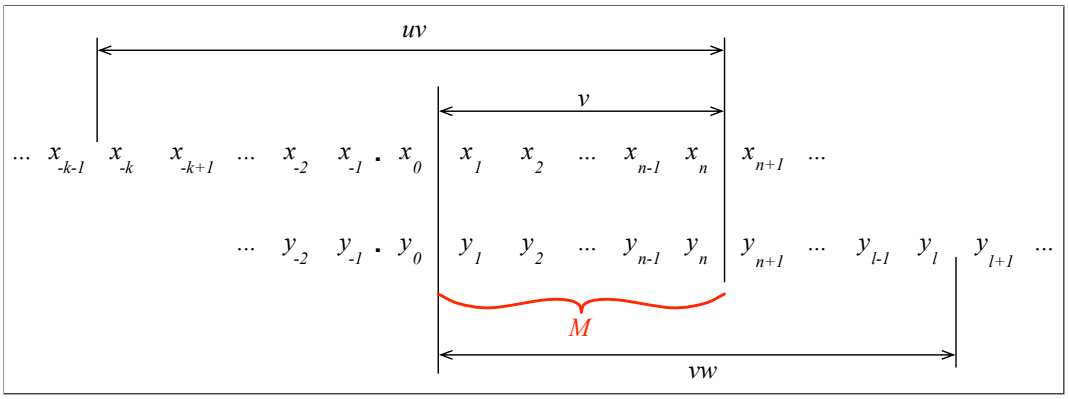


Figure 15: **Visualisation of overlap for splicing two points together,** $|v| \geq M$

Now, we claim that the point $z = x_{(-\infty,0]}vy_{[n+1,\infty)} \in X$. This is because, since we know that $v \in \mathcal{B}(X)$ by assumption, if there were a block in z that $\in \mathcal{F}$ it would necessarily have to be found in $x_{(-\infty,0]}v = x_{(-\infty,n]}$ or in $vy_{[n+1,\infty)} = y_{[1,\infty)}$, contradicting $x \in X$ or $y \in X$. This is true as long as $|v| \geq M$. The reason is not explained by Lind and Marcus probably because they consider it obvious, but it is actually a fairly subtle point. In essence, as shown in Figure 16, if $|v|$ were smaller than M, then it would be possible for a forbidden block to appear within uvw, and in particular across the $x_n y_{n+1}$ interface, because there is no guarantee that joining arbitrary symbols from different points will give an allowed block.

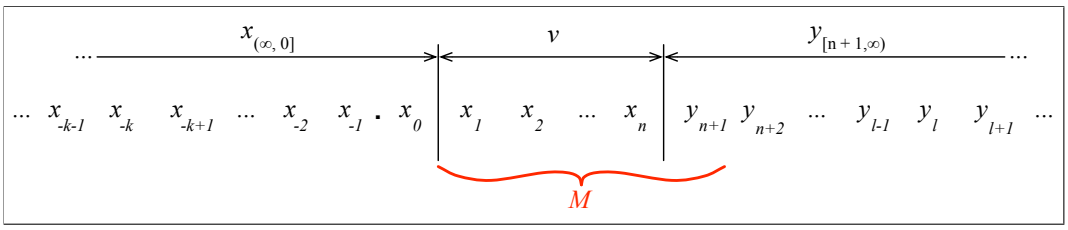


Figure 16: **Visualisation of overlap for splicing two points together,** $|v| < M$

If whenever $uv, vw \in \mathcal{B}(X)$ and $|v| \geq M$ $uvw \in \mathcal{B}(X)$, then X is M-step. More precisely, let $\mathcal{F}$ be the set of *all* $(M+1)$-blocks over $\mathcal{A}$ that $\notin \mathcal{B}_{M+1}(X)$. If these blocks $\notin \mathcal{B}_{M+1}(X)$ then as we saw in Proposition 2.9 they or any of their subsets must belong to the original set of forbidden blocks (which might have been of variable length). So if we choose M so that $(M+1)$ matches the length of the largest forbidden block in the original set, we know that $\mathcal{F}$ as defined here is a suitable set. We will show that $X = X_{\mathcal{F}}$, since that would be enough to show that X is M-step.

If $x \in X$, then no block $\in \mathcal{F}$ can occur in x, so $x \in X_{\mathcal{F}}$. Therefore $X \subseteq X_{\mathcal{F}}$.

Now let $x \in X_{\mathcal{F}}$. Then any block $\in x$ is also $\in \mathcal{B}(X)$. Thus, $x_{[0,M]}$ and $x_{[1,M+1]} \in \mathcal{B}(X)$. This is also because $\mathcal{F}$ contains only blocks of length $M+1$. Since these two blocks overlap

on M symbols, $x_{[0,M+1]} \in \mathcal{B}(X)$ too. Now, $x_{[2,M+2]} \in \mathcal{B}(X)$, and it overlaps with $x_{[0,M+1]}$ on M symbols. Thus $x_{[0,M+2]} \in \mathcal{B}(X)$. Applying this argument recursively in both directions gives $x_{[k,l]} \in \mathcal{B}(X)$, $\forall k, l \geq 0$. By Corollary 2.8.1, $x \in X$. Therefore, $X_{\mathcal{F}} \subseteq X$.
Hence, $X = X_{\mathcal{F}}$. □

Theorem 2.22 A shift space that is conjugate to a shift of finite type is itself a shift of finite type.

Proof. We refer the reader to ([62]: 31). □

2.2.7 Graphs and their adjacency matrices

A method of constructing shifts of finite type that is interesting for us starts with a finite, directed graph and generates the collection of all bi-infinite walks, i.e. sequences of edges, on the graph. It just so happens that *every* shift of finite type can be generated in this manner, i.e. as a so-called edge shift. Lind and Marcus explain that an edge shift can be understood much more easily than a general shift because we can apply powerful linear algebra techniques to its adjacency matrix. We now list some definitions we need to develop the associated ideas.

- A **graph** G consists of a finite set $\mathcal{V} = \mathcal{V}(G)$ of vertices, or states, together with finite set of $\mathcal{E} = \mathcal{E}(G)$ of edges.
- Each edge $e \in \mathcal{E}(G)$ starts at a vertex denoted by $i(e) \in \mathcal{V}(G)$ and terminates at a vertex $t(e) \in \mathcal{V}(G)$ (which could coincide with $i(e)$, creating a 'self-loop').
- We can also say that edge e has initial state $i(e)$ and terminal state $t(e)$.
- There may be more than one edge between $i(e)$ and $t(e)$ for a given e, creating a set of 'multiple edges'.
- We may use the shorthand $\mathcal{V}$ and $\mathcal{E}$ for $\mathcal{V}(G)$ and $\mathcal{E}(G)$, respectively, when the context is clear.
- For a state I, $\mathcal{E}_I = \mathcal{E}_I(G)$ denotes the set of outgoing edges, and $\mathcal{E}^I = \mathcal{E}^I(G)$ denotes the set of incoming edges.
- $|\mathcal{E}_I|$ = 'out-degree' of I, $|\mathcal{E}^I|$ = 'in-degree' of I.

Figure 17 shows a simple graph for which $\mathcal{V} = \{1, 2, 3\}$ and $\mathcal{E}$ contains 7 edges, labelled from a 3-letter alphabet (not every state transition is defined in this example if interpret edge labels as inputs).

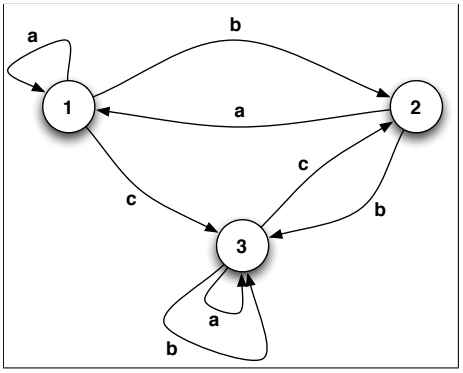


Figure 17: **Example of simple graph**

In order to define the adjacency matrix for a given graph, e.g. the example above, we need to order the vertices or states, so we follow regular numerical or alphabetical order.

Definition 2.23 Let G be a graph with set of vertices $\mathcal{V}$. For vertices $i, j \in \mathcal{V}$, let A_{ij} denote the number of edges in G with initial state i and terminal state j. Then the **adjacency matrix** of G is $A = [A_{ij}]$, and its formation from G is denoted by $A = A(G)$ or $A = A_G$. For example, for the graph of Figure 17 the adjacency matrix is:

$$A_G = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix} \tag{39}$$

Definition 2.24 Let G be a graph with edge set $\mathcal{E}$ and adjacency matrix A. The **edge shift** X_G or X_A is the shift space over the alphabet $\mathcal{A} = \mathcal{E}$ specified by:

$$X_G = X_A = \left\{ \xi = (\xi_i)_{i \in \mathbb{Z}} \in \mathcal{E}^{\mathbb{Z}} : t(\xi_i) = i(\xi_{i+1}), \forall i \in \mathbb{Z} \right\}. \tag{40}$$

The corresponding shift map over X_G or X_A is called the **edge shift map** and is denoted by σ_G or σ_A.

This definition tells us that a bi-infinite sequence $\in X$ when the terminal state of each edge equals the initial state of the next edge. So, no discontinuous jumps are allowed.

Proposition 2.25 ([62]: 36) If G is a graph with adjacency matrix A, then the associated edge shift $X_G = X_A$ is a 1-step shift of finite type.

Proof. If we list all the discontinuous jumps we will find all the forbidden blocks. Since we just need 2 symbols to describe all such forbidden combinations of edges, and since for a finite alphabet there is a finite number of possible such combinations, we necessarily obtain a 1-step shift of finite type. Following Lind and Marcus's more formal proof, let $\mathcal{A} = \mathcal{E}$ be the alphabet of X_G. Consider the finite collection of 2-blocks of $\mathcal{A}$:

$$\mathcal{F} = \{ab : a, b \in \mathcal{A}, t(a) \neq i(b)\} \tag{41}$$

Definition 2.24 shows that a point $\xi \in \mathcal{A}^{\mathbb{Z}}$ lies in X_G exactly when no block of $\mathcal{F}$ as defined here occurs in ξ. Therefore, $X_G = X_{\mathcal{F}}$, so X_G is of finite type. Since all blocks in $\mathcal{F}$ have length 2, $X_{\mathcal{F}}$ is 1-step. □

Although the shifts of finite type introduced so far are rather special, since for example they have no memory (the next state depends only on the current state, not on the previous history of the sequence), the higher block representation we discussed in Section 2.2.4 enables us to recode any shift of finite type to an edge shift as defined here.

There are several more important results about graphs, such as homomorphisms and isomorphisms, essential graphs, subgraphs, characterisation of path properties in terms of matrix properties, irreducibility, etc, but we need to press on in order to reach a high-level understanding of the potential links between symbolic dynamics and automata theory.

2.2.8 Sofic shifts

This section will unfortunately be very short, as we ran out of time. We use Williams again [97] to sum up a few important points. A **homomorphism** from one shift to another is a continuous map ϕ that commutes with the shift map. An onto homomorphism is called a **factor map** (or

quotient map). An **isomorphism** between shift spaces is also called a **conjugacy**, which we have already encountered and defined formally. We have not proven the Curtis-Hedlund-Lyndon theorem but we state it here: every homomorphism between shifts is given by a sliding block code. Another result that we have instead proven is based on the higher block shifts and states that that every shift of finite type is conjugate to a 1-step shift of finite type. This is apparently a very useful device in constructing proofs as it allows us to reduce general arguments about sliding block codes to the case of 1-block codes. In contrast to Theorem 2.22, the *homomorphic* image of a shift of finite type is not necessarily a shift of finite type. For example, the even shift is not of finite type even though it can be obtained from the golden mean shift, which is. The class of homomorphic images of shifts of finite type that *are* shifts of finite type are called **sofic shifts**.[12] Since every shift of finite type is conjugate to an edge shift, sofic shifts correspond to finite-state automata and in particular to regular languages.

We have to leave the discussion here, to be continued in the next publication on interaction computing, after the OPAALS project. After covering sofic shifts in some depth, we will attempt to connect them formally to the material in Chapter 4 on algebraic automata theory. On the strength of that connection we will seek an algebraic formalisation of algorithms through symbolic dynamics compatible with interacting automata harbouring specific algebraic structures.

Having shown how symbolic dynamics can help us analyse dynamical systems, and having introduced the p53-mdm2 system from the point of view phase space trajectories, the next chapter looks at the p53-mdm2 system experimentally. Rather than focussing on its dynamical system properties, which was already done in D1.3 [15], the work reported looks at the next step in the research on this regulatory pathway, towards a family of potential cancer drugs. Although the formal connections to this chapter and to Chapter 4 are not evident, they are not even explored, for the moment. The experimental work leads the theoretical work by 1-2 years, so that we will probably use these results in future research efforts, after OPAALS.

[12] Sofic shifts were discovered by Weiss, an Israeli mathematician, and 'Sofic' in Hebrew means finite.

3 Therapeutic Exploitation of the P53-Mdm2 Network

The OPAALS research in bio-computing, and in particular in gene expression-inspired computing, has been carried out in part as a long-term dialogue among the authors of this report. The contributions of the UNIVDUN partner[13] to WP1, WP12, and more recently to WP3 has been mainly of three kinds:

- Theoretical and philosophical work in seeking connections between natural science epistemologies and social science epistemologies.
- Numerical and mathematical work in the construction and evaluation of models for a particular regulatory pathway, the p53-mdm2 system, whose malfunction is fundamentally related to the onset of most cancers.
- Experimental cell biology work aiming to increase our understanding of this system and motivated by the applied objective of developing new cancer drugs that interact directly with this regulatory pathway.

The second and third bullet are the more relevant to gene expression computing. A discussion of the p53-mdm2 system has already been provided in [15]. The character of the work is such that experimental research tends to lead theoretical and mathematical model development. Thus, in the other chapters of this report we discuss connections between dynamical systems, symbolic dynamics, and algebraic automata theory and the p53-mdm2 system models and experimental results presented in *previous* reports and articles [14, 15, 16, 56, 21, 89]. The material discussed in this chapter, on the other hand, reports on recent experimental work that has received international recognition [57] and that has led to the identification of a potential new family of drugs, the Tenovins, but that we have not yet attempted to model mathematically.

3.1 Introduction

The *TP53* tumour suppressor gene is the most commonly mutated gene in human cancers. The p53 protein itself, which is encoded by *TP53*, is inactivated in about 50% of adult solid tumours, whereas in the remaining 50% the function of p53 is totally or partially impaired by modifications in other proteins within the p53 pathway. In order to explain p53's biological function and the therapeutic implications thereof, we will first introduce some key concepts and terminology (summarised in Table 1) related to the process of protein synthesis. The chemical composition of each protein is encoded by a specific gene, of which one or more copies can exist in the genome. During the first step of the synthesis of a protein, known as *transcription*, the gene sequence is 'read' and copied in the form of a *messenger RNA* sequence (mRNA). This process is tightly modulated, for instance by the binding or so-called *transcription factors* to the *promoter region* of the gene. Next, some fragments of the mRNA molecule can be 'reorganised', which means that a single gene can encode for more than one kind of protein. Then, the mRNA sequence is 'translated' once again, this time into the amino-acid sequence that constitutes the basic chemical composition of the protein. Finally, this amino-acid sequence is generally 'edited', by chemical changes (i.e. *post-translational modifications*), and 'folded' in three-dimensions to give rise to a fully functional protein.

[13]The UNIVDUN team is composed of cancer biologists and cancer doctors from the University of Dundee, some of whom (Sonia Lain and Inge Van Leeuwen) have now moved to Karolinska Institutet in Sweden.

PROCESS	KEY PLAYERS	DESCRIPTION
Transcription	Genes, Promoters, RNA-polymerases, Transcription factors	RNA-polymerases 'read' the DNA code of a gene and produce a complementary mRNA sequence. Transcription factors, which are proteins that bind to the promoter region of specific target genes, regulate the activity of the RNA-polymerases.
Splicing	Spliceosome, Introns, Exons	A protein complex known as spliceosome cuts noncoding fragments (introns) out of the original mRNA sequence and combines remaining fragments (exons) into a final mRNA molecule
Translation	Ribosomes	Ribosomes 'read' the mRNA sequence and use it to synthesise a specific amino-acid chain.
Folding & Post-translational modifications	Chaperones	Amino-acid chains generally need to fold, as well as undergo various chemical modifications, in order to give rise to a fully functional 3D protein structure. The folding process can be assisted by proteins known as chaperones.

Table 1: **The basics of protein synthesis reminded**

p53 executes its anti-cancer function by acting as a stress-inducible transcription factor[14] as follows. In response to a variety of potentially carcinogenic stimuli, such as exposure to radiation or DNA-damaging compounds, the p53 protein undergoes post-translational modifications[15] and becomes 'active' [63, 85, 88]. This more stable and transcriptionally active form then induces the synthesis of a large number of target proteins, which carry out DNA repair, reversible cell-cycle arrest, irreversible cellular senescence or, when appropriate, programmed cell death [78]. The function of p53 at the molecular level is summarised in Figure 18.

As p53 governs over the life and death of cells, it is not surprising that several layers of regulation tightly control its activity. The main regulator is mdm2 [71], an enzyme that keeps p53 levels low in the absence of stress [46, 71]. Importantly, p53 and mdm2 are also linked by a transcriptional feedback-loop (Fig. 19), as *MDM2* is a known p53 target gene [68, 98], that can cause p53 levels to oscillate in response to stress [4, 9, 15, 43, 55, 89, 61, 72]. Interestingly, in tumours with mutant p53, this feedback loop is absent, as the faulty p53 molecules are unable to act as transcription factors and therefore mdm2 levels remain very low. Consequently, cancer cells with mutant p53 often present very high levels of the tumour suppressor.

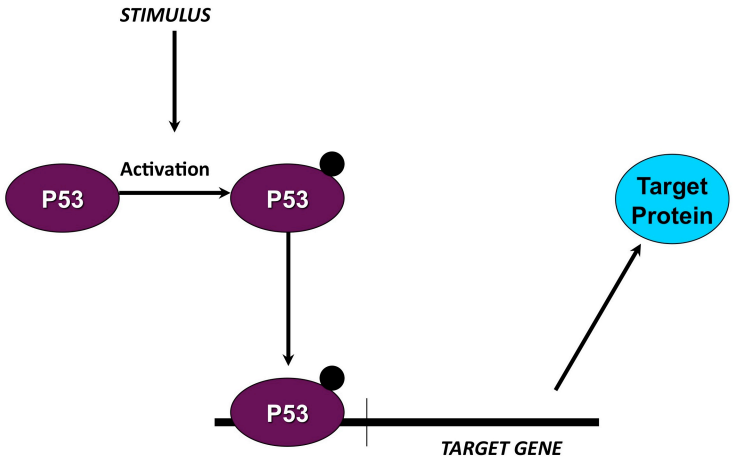


Figure 18: **Simple schematic of p53's function as a transcription factor. The p53 tumour suppressor is activated in response to various hazardous stimuli, such as radiation, oxidative stress and viral infections. Active p53 then enters the nucleus and regulates the expression of over a hundred target genes [34, 96].**

[14]Concept defined in Table 1
[15]Concept defined in Table 1

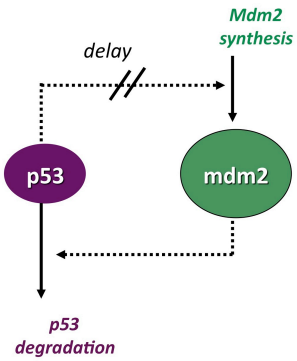


Figure 19: **Negative feedback loop linking p53 and mdm2. The p53 tumour suppressor promotes the synthesis of mdm2 [68, 98], while mdm2 proteins bind to p53 and label it for degradation [46, 71]. As protein synthesis (Table 1) is a relatively slow molecular process, there is a so-called *transcriptional delay* in the feedback system. Both p53 and mdm2 are highly-regulated proteins [54].**

3.2 P53-based drug screening

P53, also known as the *guardian of the genome* [58], is arguably the one of most important known tumour suppressor in humans. In mice, transgenic animals lacking functional p53 present a substantial increase in tumour incidence compared to wild-type animals [10, 45, 60]. Moreover, restoration of p53 function has been shown to induce growth inhibition in cultured cancer cell lines [37, 86] and to cause regression of lymphomas and sarcomas in mice [95]. These observations make p53 a highly-attractive target for cancer therapy [8, 59]. There are currently two main lines of research regarding the therapeutic exploitation of p53 that aim at (1) 'repair' dysfunctional p53 molecules in cancers with mutant p53, or (2) increase p53 levels in cancers with wild-type p53. Here we will focus on the latter kind of approach.

Large libraries of chemical compounds are now commercially available, which makes it possible to perform high-throughput screens to identify potential new anti-cancer drugs. One possible screening approach is to seek compounds that interfere with a specific biochemical reaction. In very simple terms, the involved proteins are purified, put in a test tube in suitable conditions, and then exposed to each compound in the library. This is known as *reverse chemical genetic* (RCG) approach or *biochemical screening* approach. In the case of the p53 pathway, for instance, an obvious way to try to enhance p53's activity is to impede mdm2's down-regulating function [91, 92]. A chemical screen for p53-activators based on this principle led to the successful discovery of the nutlins, selective small-molecule activators of p53 [93]. The nutlins have been shown to be capable of inducing up to 100% growth inhibition in various tumour models in mice.

The RCG approach above has some intrinsic limitations, including the following. Firstly, it requires that the target, for example the mdm2–p53 interaction, is chosen beforehand. Secondly, it has to be possible to purify all components involved in sufficient amounts for the screen and to get the reaction of interest taking place in a test tube. Thirdly, a compound that constitutes a hit in the test-tube assay might not work in cells. It could, for instance, be highly toxic, incapable of entering the cell or rapidly degraded. An alternative way of screening compounds is to seek for a desired biological effect, such as the activation of the p53 tumour suppressor, without worrying beforehand about *how* this effect occurs. This approach is known as *forward chemical genetic* (FCG) approach, or *cell-based screen*, and has the great advantage that hit compounds emerging from the screen are known to work in cells and are non-toxic. The main inconvenience or challenge associated with this approach is, however, that the mechanisms of action of the hit compounds are unknown. For example, a hit compound from a screen for p53 activators could act directly on p53 itself or on any other protein modulating p53's activity. Recently, a p53-based FCG approach has lead to the successful discovery of the tenovins [57], a family of small-molecule activators of p53, and the JJ78 family [87], novel tubulin poisons with clear anti-tumour activities. The former molecules constitute the subject of the next section.

3.3 Discovery & characterisation of the Tenovins

In 2008, Lain and co-workers [57, 67] screened a small-molecule library for p53 activators. For this purpose, they seeded cells onto 96-well plates and exposed them to the different compounds in the library. To facilitate the detection of changes in p53 activity, the authors chose for their study a genetically-modified mouse cell line that expresses the *lacZ* gene, coding for a protein called β-galactosidase (β-gal), under a p53-dependent promoter[16]. This means that the β-gal enzyme is only present when p53 is acting as a transcription factor. Importantly, β-gal activity can be readily measured via a simple CPRG test, as CPRG cleaved by β-gal gives a pink product that can be quantified by spectrophotometry. That is, expressed in lay terms, the p53-activating ability of chemical compounds can be evaluated based on the appearance of a pink colouring. This is illustrated in Fig. 20A for actinomycin-D, a known activator of the p53 tumour suppressor. In the absence of actinomycin-D the (control and DMSO) wells present a yellow colouring, which corresponds to uncleaved CPRG, whereas in the presence of actinomycin-D the wells acquire a pink colouring. Beyond a concentration of 50nM, however, actinomycin is toxic to the cells and, consequently, there is less β-gal activity and thus less pink product. Fig. 20B shows a plate from an actual drug screen based on the same technique.

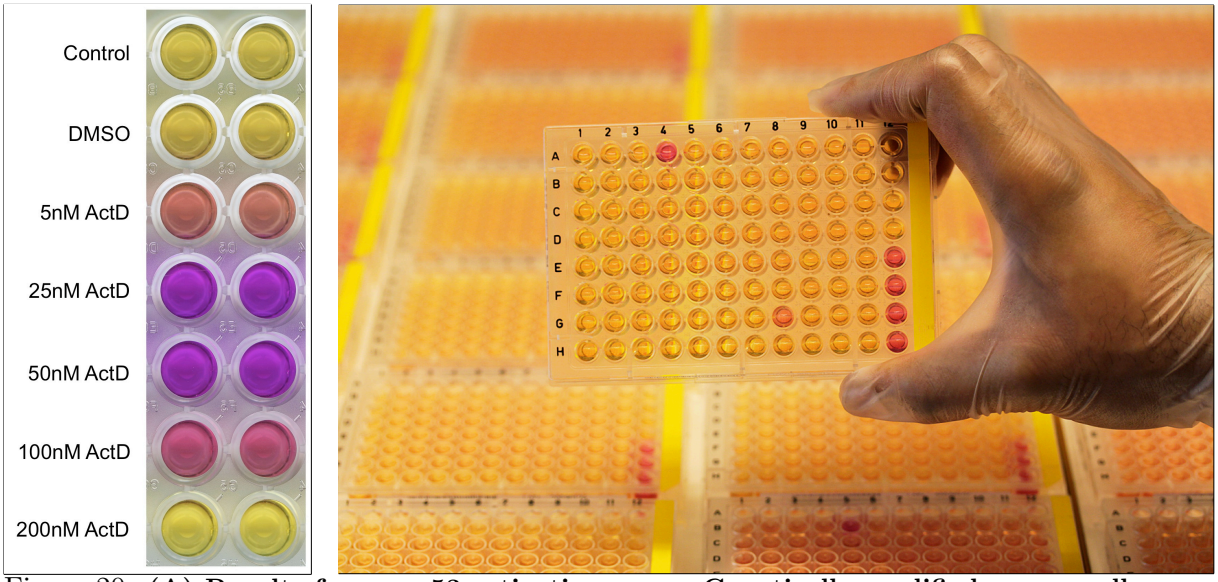


Figure 20: **(A) Results from a p53-activation assay. Genetically-modified mouse cells were seeded on a 96-well plate, left two grow for two days, and then treated with different doses of actinomycin-D (two wells per dose). The presence of p53-activity was revealed by the emergence of a pink colouring following incubation with CPRG mix (for detailed experimental protocol, see [57]). Maximum p53 activity was achieved at ActD 25nM and 50nM. Note that the intensity of the colouring starts to decrease at doses beyond 50nM, as then actinomycin-D becomes increasingly toxic for the cells.**
(B) Application of the same technique during a drug screen. Each well in columns 1 to 11 was treated with a different compound from a chemical library. In column 12, the first 4 wells were exposed to a negative control (DMSO), whereas the remaining were treated with a positive control (20nM ActinomycinD). The highlighted plate reveals two clear-hit compounds, i.e. substances that activate the p53 tumour suppressor, in wells A4 and G8.

A screen of 34 000 small-molecules based on the primary cell assay shown in Fig. 20 yielded an initial hit rate of about 2%. These 623 compounds were further tested (for technical details, see Supplemental Data in [57]), for instance using different concentrations, and prioritized according to their performance. These secondary assays led to a final pool of 31 compounds of interest. In

[16]Concept defined in Table expression

what follows, we will focus on the characterisation of one of them, which was coined tenovin-1. In cell culture, Western Blot analyses[17] demonstrated that this compound is able to increase p53 protein levels substantially in less than two hours exposure. Moreover, both protein and mRNA[18] levels were increased for various known p53 targets, such as the p21 protein, meaning that p53 is transcriptionally active [57]. The levels of p53 mRNA, however, were the same in untreated and tenovin-treated cells, which suggests that this compound increases p53 levels not by promoting its synthesis but by enhancing its stability.

Next, tenovin-1's anti-tumour potential was tested on several cancer cell lines. Long-term treatment (4 days) induced cell death and decreased cell growth in all of them [57]. The ARN8 melanoma cell line, which was particularly sensitive to tenovin-1, was selected for xenograft studies, where human cancer cells are implanted onto mice that have a deficient immune system. Tenovin-1's water solubility, however, turned out to be too low for use *in vivo*. Therefore, collaborators at the School of Chemistry at the University of St. Andrews modified the chemical structure of tenovin-1 by adding a solubilizing group. The resulting compound, tenovin-6, not only conserves tenovin-1's p53-activating properties, but even is slightly more effective. Importantly, the growth of ARN8-derived tumour xenografts was significantly reduced in tenovin-6 treated mice [57]. These promising observations convinced Lain *et al* (2008) that is would be well-worth trying to identify the mechanism of action of tenovin-6.

The sensitivity of a cell to chemical compounds depends on how much of the target protein is present which, in its turn, depends on the number of copies of the gene encoding for it. For instance, lets consider a compound X that inhibits the activity of an enzyme-Y. Then cells bearing only one copy of the Y gene, instead as the usual two copies, are likely to present lower levels of Y, which possibly renders them hypersensitive to X. Hence, in order to identify the molecular target of tenovin-6, Lain *et al.* [57] used a collection of more than 6000 genetically-modified yeast strains, each of which has one copy of a particular gene missing. Among those strains that were most sensitive to tenovin-6 was the one carrying a *SIR2* deletion. This was a very interesting discovery, as SIR2 is the yeast homolog of mammalian proteins called sirtuins, which are involved in both ageing and cancer [90]. Furthermore, one of the sirtuins, SirT1 is known to carry out post-translational modifications[19] of various substrates, including the p53 tumour suppressor [94]. It was thus well possible that tenovin-6 affects p53 activity by targeting SirT1. Indeed, two mammalian sirtuins, SirT1 and SirT2, were confirmed as targets of tenovin-6 in *in vitro* biochemical assays using purified proteins.

Figure 21 summarises the interplay between p53, mdm2, sirtuins and tenovin-6. In the absence of stress, mdm2 binds to p53 and labels it for degradation by adding so-called ubiquitin groups to p53's structure. As a consequence of this mdm2-mediated modification, p53 becomes less likely to undergo an alternative post-translational modification[20] consisting of the addition of acetyl groups. The levels of acetylated p53 are down-regulated further by SirT1, which is an enzyme that removes acetyl groups from its substrates. In the presence of stress, p53 becomes acetylated, which increases its stability, by preventing mdm2-mediated ubiquitination, and enhances its activity as a transcription factor [63, 88]. Finally, tenovin-6 promotes p53 acetylation and stability by inhibiting SirT1's deacetylase function. That is, tenovin-6 affects the behaviour of the p53 pathway as a dynamical system. In the absence of stress, the reactions (represented by arrows) tend towards the left of in panel 21.A. In this context, p53 is preferentially degraded and, in particular, does not induce the synthesis of target proteins, including mdm2. Consequently, the levels of both p53 and mdm2 remain low. In contrast, in the presence of an oncogenic

[17] A method to quantify the total amounts of specific proteins in a cell population
[18] Concept defined in Table 1
[19] Concept defined in Table 1
[20] Concept defined in Table 1

stimulus, p53 becomes protected against mdm2 and, therefore, the reactions in panel 21.B tend towards the right. That is, in this context, p53's tumour function as a transcription factor is being promoted. Finally, exposure to tenovin-6 has comparable effects, as it prevents the inactivation of active p53, thereby causing accumulation of p53 in its active form and the expression of target genes.

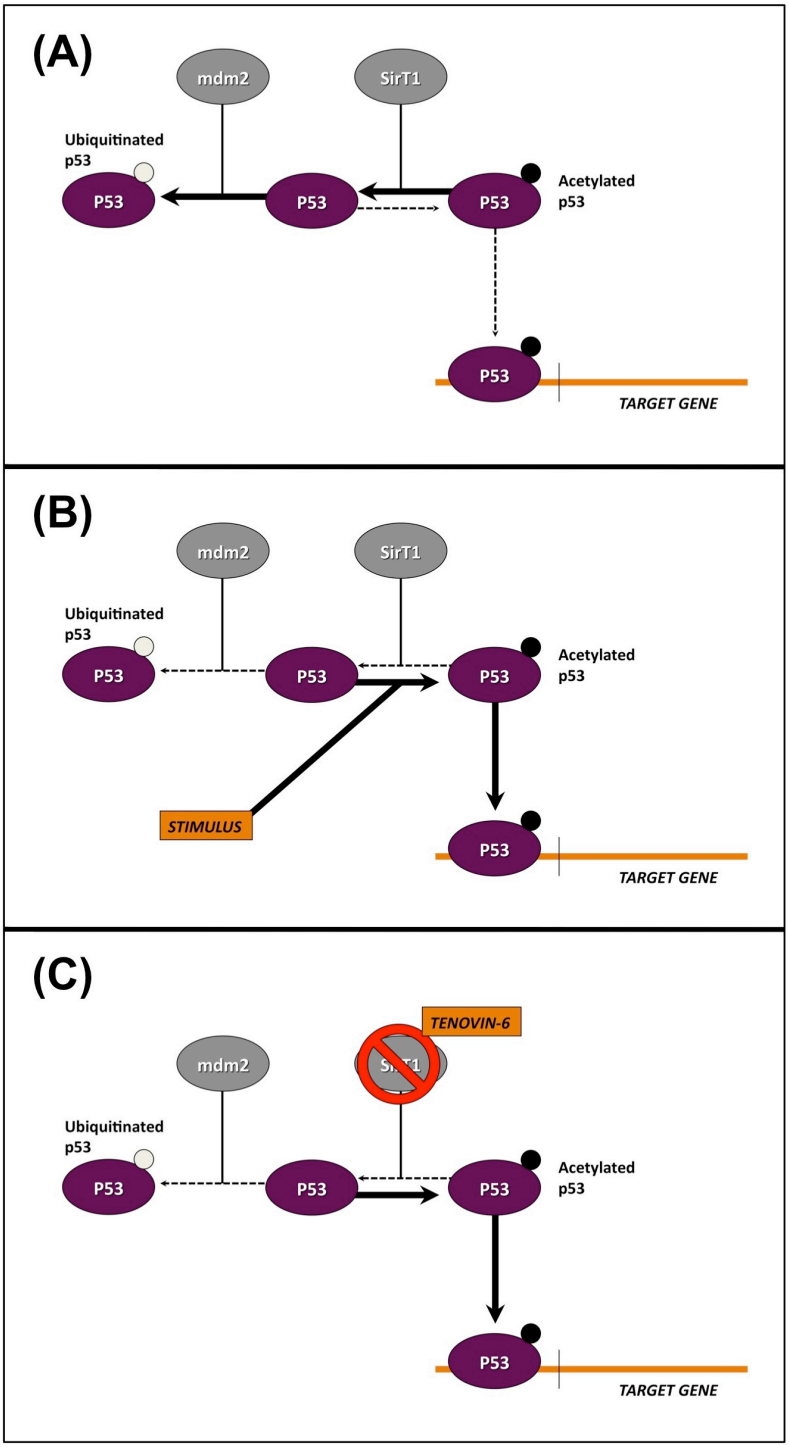


Figure 21: **Schematic summarising the mechanism of action of the tenovins. (A) In the absence of stress, mdm2 adds ubiquitin groups to the p53 molecule, targeting it for degradation. SirT1 removes acetyl groups from any 'active' p53 molecules [94], which become inactive and vulnerable to mdm2-medited degradation. In sum, under this conditions, p53 is unstable, its levels remain very low, and the p53-target genes are not expressed. (B) In the presence of stress, p53 undergoes chemical modifications, such as acetylation and phosphorylation, that enhance its performance as a transcription factor and also prevent downregulation by mdm2 [63, 85, 88]. In sum, under these conditions, p53 is both stable and active, and the p53-target genes are expressed. (C) Tenovin-6 inhibits SirT1 function [57], thereby promoting the accumulation of acetylated p53 and thus indirectly interfering with mdm2s downregulating function. In sum, tnv6 mimics the presence of stress by increasing the levels of active p53, leading to the expression of p53 target genes.**

Having gained greater familiarity with the p53-mdm2 system, in the next chapter we go back to an algebraic perspective, and in particular to algebraic automata theory. At the end of the chapter we apply the computational algebra tools developed in Year 4 of OPAALS to the p53-mdm2 system.

4 Algebraic Structure Analysis of Dynamical Systems

The work reported in this chapter builds on approximately 50 years of research in algebraic automata theory, some of which was performed by researchers who founded the field and who are still actively collaborating with the UH team. We can only skim the surface of this huge field in this short chapter, but we have taken pains to make the concepts accessible and to highlight the contributions provided during the final year of the OPAALS project.

In spite of its overall accessibility, the material discussed in this chapter is very advanced, and can therefore benefit from some context-setting remarks. In particular, in the DBE, BIONETS, and OPAALS bio-computing research activities we have always emphasised the importance of symmetries in the search for a model of computation that can construct order automatically, through interacting software artefacts. A symmetry is an invertible transformation that leaves some feature of a mathematical object invariant. Because the set of invertible transformations of a mathematical object that leave some feature of that object invariant always form a group, we have always stressed that, whatever form the solution to the bio-computing problem might take, it would have to have a strong link to algebra.

The discrete nature of digital systems has motivated us to keep pushing for a bridge between dynamical models of biological behaviour based on continuous mathematics, on the one hand, and behaviour specification languages for software engineering, on the other. The 'stuff' of the bridge itself appears to require the integration of an algorithmic or behavioural perspective with a state-set or structural perspective, both cast in discrete-mathematical or algebraic language. The symbolic dynamics discussion of Chapter 2 addresses the former, whereas this chapter addresses the latter.

4.1 Introduction

4.1.1 Basic concepts

Readers who are unfamiliar with discrete group theory can find explanations of some of the elementary concepts in previous BIONETS and OPAALS reports and articles [22, 16, 20, 21, 56, 83, 24]. To begin our discussion of algebraic automata theory we need to understand the concept of transformation semigroup. This is most easily done by analogy with groups, as shown in Figure 22. Transformation semigroups are the mathematical objects that model automata. We are only concerned with the finite variety, meaning finite semigroups operating on finite sets of states [81].

Figure 22: **Definition of basic terms**

It is helpful to fix these basic ideas with a couple of pictures. Figure 23 (which reproduces Figure 12 of [22]) shows a comparison between an abstract group and a permutation group, while Figure 24 (reproducing Figure 49 of [83]) gives a sense of what is involved with semigroups.

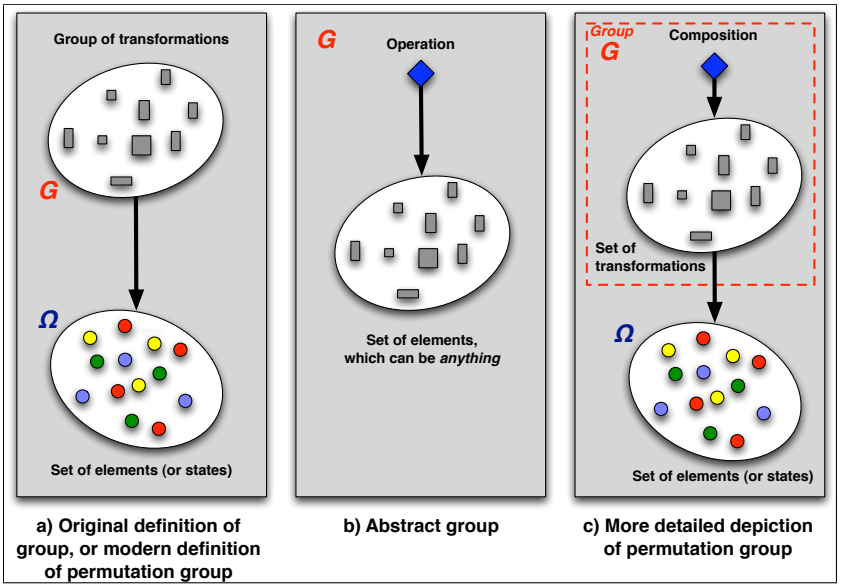


Figure 23: **Comparison of abstract group with permutation group**

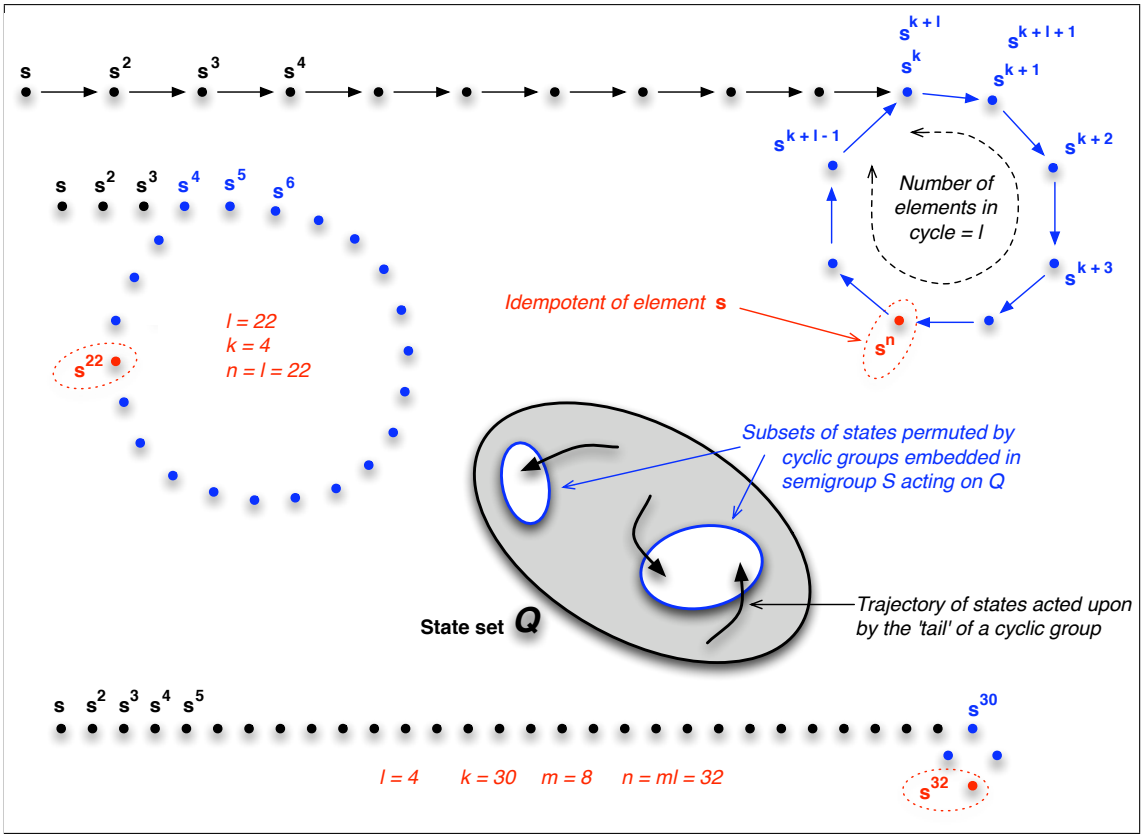


Figure 24: **Visualisation of different possible collections of semigroup elements, showing how cyclic groups can be embedded in a semigroup. Also shown is the set of states Q being acted upon, with a few representative state trajectories. Black dots: non-invertible elements; blue and red dots: invertible elements. For a more detailed discussion, see [83].**

If S is a semigroup, the element $s \in S$ is **idempotent** iff $s^2 = s$. The concept of idempotent can be seen as a generalisation for semigroups of the identity element for groups, although semigroups can have, independently, also an identity element. Any kind of group, not just cyclic groups as depicted in the figure, can be found in a semigroup. In particular, the simple non-abelian groups (SNAGs) are very interesting from the point of view of functional completeness and alternatives to Boolean algebra as a basis for computation [50]. After these preliminary remarks we can start the discussion of the algebraic structure analysis of dynamical systems.

4.1.2 Motivation

Our ultimate goal is to understand and to engineer dynamical systems. By 'engineer' we mean the ability to predict and manipulate their behaviour, i.e. to influence their dynamics in order to reach some target state of the system, or some target behaviour (e.g. a desired limit cycle). We can view dynamical systems from different perspectives but, once we handle them as discrete state transition systems, then collapsings of states or cycles in state space and their intricate interrelations become the primary objects of study. Therefore algebraic automata theory becomes a useful tool for identifying subsystems and discovering their relationships, and for formulating this knowledge in a way that supports creative manipulation. Briefly, we call these activities a *coordinatization*.

Hierarchical coordinate systems can obviously be used for dynamical systems, but not directly. As shown in Figure 25, which builds on Figure 5, we have to go through several intermediate steps in order to get an automaton description of the given dynamical system. These intermediate steps may involve technical difficulties (e.g. increasing the size of the state space, or there may be different alternatives for doing one step that possibly yield radically different coordinate systems, as in a Petri net-to-finite-state-automaton conversion [29]). Therefore, the long-term goal is to find the ways in which hierarchical coordinates can be used directly for dynamical systems. For working towards this goal, we need to make conceptual and technical advances in dealing with hierarchical coordinatizations.

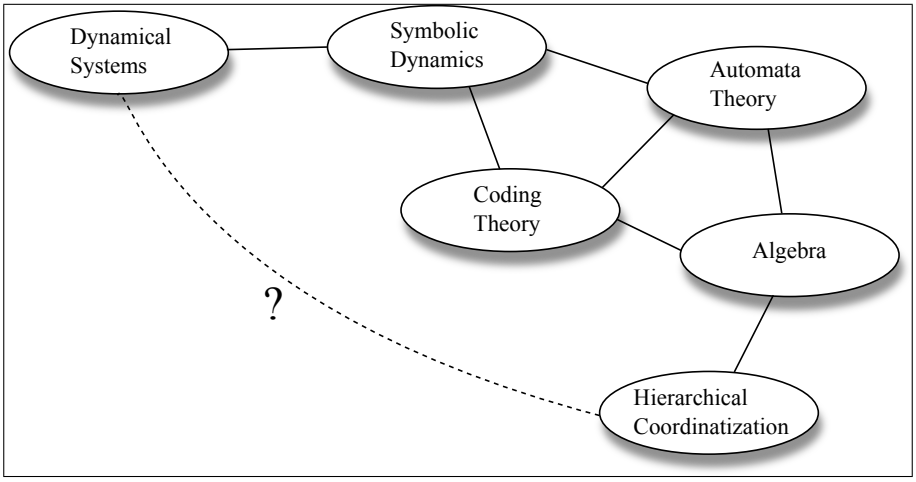


Figure 25: **Finding the shortcut for applying the findings of algebraic coordinatization to dynamical systems**

One of the biggest achievements here is the development of `SgpDec`, the open-source algebraic software package for cascaded structures and group and semigroup decompositions. This package is written for the `GAP` open-source computer algebra system which currently contains the most advanced and most efficient computational group theory algorithms. The current version of `SgpDec` is the culmination of several years of research and software development. During the

OPAALS project its usability and scalability improved to a level that opens up a whole range of new possibilities in applied algebraic automata theory.

Constructive mathematical proofs are basically algorithms, but mathematical proof may neglect computational feasibility issues. The holonomy proof [99, 100, 33, 41, 48, 23] of Krohn and Rhodes's prime decomposition theorem for finite semigroups and machines [53] is the closest one to a computer scientist's way of thinking. The holonomy coordinate system arises from a detailed analysis of how the semigroup acts on those subsets of the state set that are images of the state set under the action of the semigroup.[21] Roughly speaking, a new level in the hierarchy appears when there is some irreversible step in the dynamics (e.g. collapsing two or more states). Symmetries may appear when there is a set of subsets of the state set that are mutually reachable from each other. The existence of these local pools of reversibility is a necessary but not sufficient condition for the existence of nontrivial group components. Symmetries are periodic reversibility operations rather than just resets, thus the semigroup should act on these local pools in a special way (capable of periodic returns in a nontrivial cycle) in order to form group components. For more details see [28].

4.1.3 Algebraic Automata Theory

The hierarchical composition (cascade product) is a proper balance between two conflicting requirements: having a nice, comprehensible structure and possessing the expressive power to construct any arbitrary automaton. The parallel composition (direct product) has a very simple structure, all of its components are completely independent, but this also means that it is not possible to surpass the complexity of the building blocks. On the other hand, if we allow arbitrary wiring of the components (feedback loops with different lengths) then any system is realizable (even using only flip-flops as building blocks – see, e.g. [23]), but such a construction generally provides no insight into structure or dynamics. In contrast, with a cascaded decomposition, feedbacks in the original automaton being decomposed can give rise to structural permutation groups – possibly at different levels in the hierarchical decomposition – reflecting local symmetries in the system (i.e. acting on a subset of the state set). An easy example would be a set of identical components with two states $\{0, 1\}$ put into a directed loop (in other words the components form a line but the last element has a feedback to the first one) and a global operation that locally resets the state to 1 if the predecessor is in state 1, otherwise the updated state is 0. This operation applied to the state vector $(1, 0, \ldots, 0)$ repeatedly creates a cyclic permutation.

For explaining Krohn-Rhodes Theory, the best way is to present it by using a metaphor. Basically we do the same thing that the decomposition into prime factors does for the integer numbers, but instead of integers we do it for more complicated structures, namely finite-state automata considered as transformation semigroups (see Figure 26).

The basic building blocks are (1) the simple[22] permutation groups (for the reversible computation) and (2) a single additional building block for the irreversible computation, the so-called *flip-flop automaton*, which is essentially a one-bit resettable memory that can be set and read.[23] The simple groups are called the '*primes*' of this theory, as they play a role analogous to the

[21] Also included in what the semigroup acts upon are the singletons and whole state set, whether or not they are images.

[22] This has a well-defined meaning in group theory: a group is *simple* if it has only trivial homomorphic images, i.e. any structure preserving map to another group is either one-to-one or collapses all elements to a single point. Alternatively, a group is simple if it has no normal subgroups. See, e.g. [82].

[23] An important but subtle point here is that although the flip-flop can be reset, this does *not* make it reversible.

prime numbers in the multiplicative decomposition of integer numbers (see Figure 26 and the discussion of this analogy below the table).

	Integers	**Finite Automata**
Building Blocks	Units (± 1), Primes	Flip-flop Automaton, Permutation Automata
Composition	Multiplication	Wreath Product
Precision	Equality	Division, Emulation
Uniqueness	Unique (up to units)	Different Decompositions

Figure 26: **The analogy between the factorization of integer numbers and finite state automata. The very same idea is applied to computational structures in algebraic automata theory. Here we have two types of irreducible building blocks (also called** *primes* **(simple groups) and** *units* **(divisors of the flip-flop)), but as automata are more complicated structures, we have** *emulation* **of the decomposed structure by possibly larger, decompositional structure (also called** *division* **of the decomposition by the original automaton) rather than equality, and the factorization need not be unique at all.**

We should qualify this explanation somewhat. Whereas the Krohn-Rhodes theorem does break down a semigroup into trivial and simple groups and flip-flops, the holonomy decomposition (to be further discussed below) breaks a semigroup down only to permutation groups and flip-flops. So from the practical point of view of the holonomy decomposition the basic building blocks are permutation groups, not simple groups. A given permutation group may contain simple groups, and this further decomposition can be effected as a Lagrange group decomposition, also implemented in [32]. Finally, each simple group may occur on its own rather than inside another permutation group, in which case the holonomy result coincides with the Krohn-Rhodes theorem.

It is an interesting question why algebraic decompositions disappeared from mainstream computer science as taught to undergraduates. For instance, a textbook from the early 80s still had a chapter on cascaded connections of automata [40, Ch. 8], but subsequent editions removed that content. Why has this happened instead of simplifying and streamlining the theory to make it easier to teach and learn? A possible explanation is the lack of driving applications at that time, which in turn can be explained by a lack of computational implementations (see Section 4.4.1).

4.2 Lagrange and semigroup decomposition, and wreath product

We need to give at least a cursory intuitive description of the wreath product, otherwise the discussion that follows will be difficult to understand. We do this by analogy with the decomposition of groups. Every finite group can be decomposed into a finite 'composition series' of nested subgroups, where each is a largest normal subgroup of the subgroup immediately above it. In the case of A_4 such a series is: $A_4 \trianglerighteq V_4 \trianglerighteq C_2 \trianglerighteq \{e\}$. Figure 27 shows the relationships between these groups, where A_4 in this case is taken to be the rotational symmetry group of the tetrahedron, which we have extensively studied in previous reports [22, 16, 83, 21]. When a subgroup N of a given group G is normal, it and its cosets form the elements of a new group, called the factor group or quotient group and denoted by G/N. By Lagrange's theorem, the size of the factor group is $|G|/|N|$. The ratios of successive members of the composition series are simple groups. In Figure 27 we show that $A_4/V_4 = C_3$, and we show how the 12 elements of A_4 are grouped into 3 cosets, which (by Cayley's theorem, see [16, 21]) are analogous to 3

Indeed, it is not possible to reverse a resetting operation since this erases the previous state, and hence is not a permutation of the flip-flop's state set.

'macro-states'. The next factor, $V_4/C_2 = C_2$, ought to be shown as just two macro-states or cosets, $\{e, 4\}$ and $\{9, 11\}$, but we are showing instead how the *original* group A_4 gets partitioned into 6 cosets. These 2 are in any case depicted as well. Finally, the Lagrange decomposition is unique up to isomorphism.

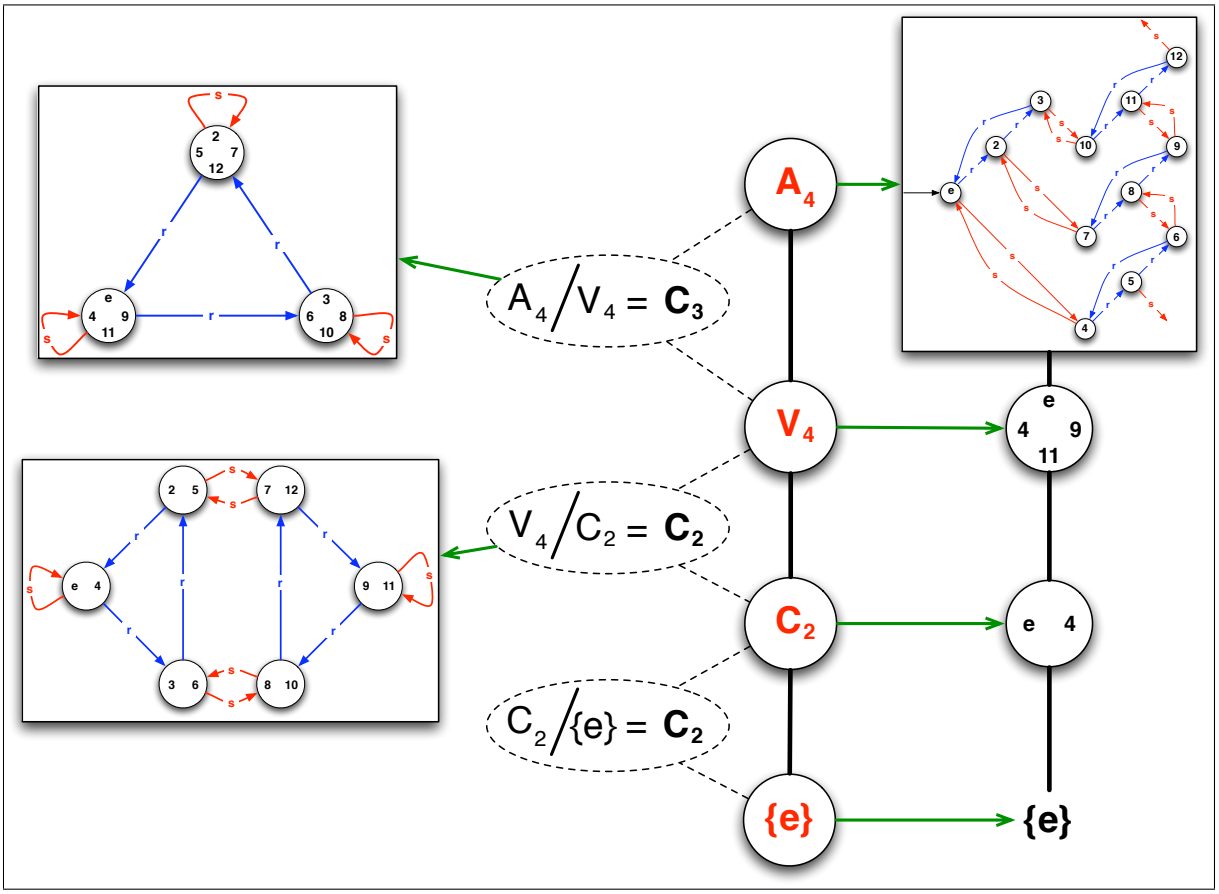


Figure 27: **Visualisation of the Lagrange decomposition of** A_4

Figure 28 shows a hypothetical wreath product. The ovals indicate semigroups at different levels in the decomposition, whereas the squares indicate the state sets present at the corresponding levels. At any one level, a particular semigroup element (indicated by a letter) operates on the set of states at its same level, shown as a red arrow inducing the state transitions shown as green arrows. However, *which* semigroup elements operate depends on the states of the levels above through so-called dependency functions. The idea of the dependency functions is that when a state transitions to another state at a given level (in response to e.g. a suitable input) this state transition may trigger the action of a semigroup element at the levels below. These triggers are shown as dotted blue arrows. Similarly, the trigger on a semigroup element at a given level may depend on state transitions happening at *all* the levels above, and not just at the level immediately above. In order to be able to define a function, the latter perspective is adopted (as the former would not be single-valued).

In this example, we can recognise a cyclic group operating on 3 states at the second level (C_3), and four C_2 cyclic groups operating on the bottom level. This example is totally arbitrary and only intended to provide a visualisation of the wreath product, so it's not clear what automaton it corresponds to, if any. In the discussion that follows the letters X or Q denote the state set, whereas the semigroup of transformations acting on the state set is denoted by S. Thus, a transformation semigroup is the ordered pair (X, S).

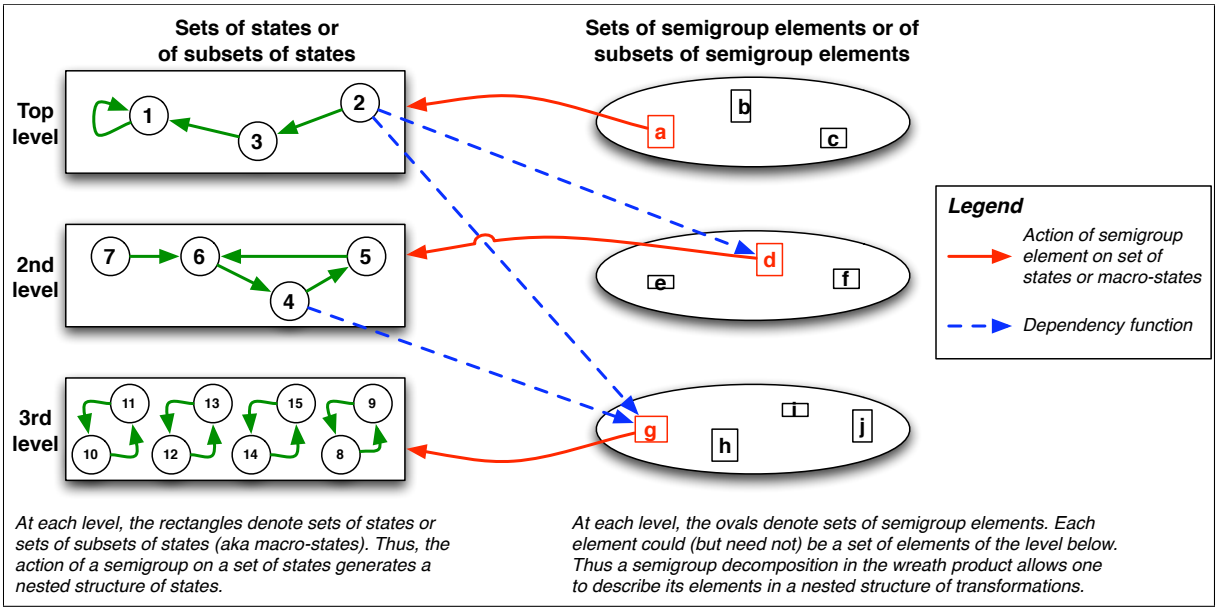


Figure 28: **Visualisation of an element of a wreath product. The wreath product formalises the interaction between nested structures.**

4.3 Conceptual advances

Decomposition theories usually provide an embedding of the original structure into a potentially bigger composite structure. In the case of hierarchical decompositions the composite structure is a wreath product, which is in a sense a complete structure. Due to its size the wreath product can easily become intractable. For instance, we can use the wreath product for coordinatizing C_4, but we get a coordinate system of something bigger: $C_2 \wr C_2 \cong D_8$, which is the group of symmetries of the square. The embedding obviously works $C_4 \hookrightarrow D_8$, but the dihedral group has the flip-symmetry as well, beyond rotations. The discrepancy is relatively big and it gets bigger for groups with more elements. Therefore we aim to have more 'economical' coordinate systems by using sub-wreath products.[24] This is a slight change in viewpoint as decomposition theories normally consider embedding into the wreath product (which as we said is usually a very big structure), but here we would like to put together the components more efficiently, thus we study 'sub-wreath' products, that we call *cascaded structures*.

The problems are:

1. Wreath products have difficult definitions mostly in abstract algebraic terms.
2. Wreath products are huge structures, therefore computing with them is usually not possible due to combinatorial explosion.
3. Due to their complexity, wreath products are difficult to grasp (except in some special cases) and thus they do not contribute to our understanding of the original automaton being modelled as a cascade.

We suggest the following solutions:

1. Explicitly building the unidirectional connection network (dependency functions) between the components.

[24] It is interesting that in the case of groups we can construct an isomorphic coordinate system, whereas this is not the case for semigroups.

2. Dealing with sub-wreath products.
3. Simplified visual notation for cascaded automata.

4.3.1 Dependency functions

In order to build cascaded structures with smaller combinatorial footprints we would like to reduce the amount of wiring in the hierarchical structure. To achieve this here we define the connections between the components precisely. We fix a list of components $L = \big((X_1, S_1), \ldots, (X_n, S_n)\big)$ where the indices correspond to depth values indicating the hierarchical level, then

Definition 4.1 A *dependency function* d_i in L (of level i) is a function

$$d_i \colon X_1 \times \ldots \times X_{i-1} \to S_i \quad i \in \{1 \ldots n\}. \tag{42}$$

Thus a dependency function of level i has arity $(i-1)$. This includes the $i = 1$ case as well, when the dependency can be considered as a constant, a function with no arguments: $\{1\} \to S_1$. The intuitive interpretation of the dependency function is that it provides a set of rules with which one can pick an actual operation from a component based on the states of the components above (an easily understandable example is provided in Section 4.3.3).

The wreath product contains all dependency functions for each level and all of their possible combinations, hence it keeps exploding combinatorially. However, constructive algorithms for decompositions build 'economical' coordinate systems by using sub-wreath products, and in the case of groups we can even construct an isomorphic coordinate system. Therefore, focusing on dependency functions and treating them as first class objects in implementations are important conceptual steps.

4.3.2 Evolving and using new notations

In mathematics notation is an important issue. It can promote or hinder the understanding of ideas. Therefore we put special emphasis on choosing and developing the right notation.

Transformations of Finite Sets From early on, group theory had its specialized notation for permutations, the cyclic notation, as it is easy to observe that a permutation consists of distinct cycles. On the other hand semigroup theory seems to be more reluctant to depart from the conventional 2-line matrix-like notation, which is rather space-consuming and makes recognizing the salient features of transformations rather difficult. A new notation was introduced in 2005 [3], then a more refined one in 2009 [38, Ch. 1.3]. We use the latter one with slight modifications in order to obtain an even more compact notation.

Considering the mappings as digraphs, each transformation consists of one or more components. Each component contains a cycle (possibly a trivial cycle). Unlike the permutation case, the points in the cycle can have incoming edges, denoted by

$$[\ \text{source}_1, \ldots, \text{source}_m\ ;\ \text{target}\ ]$$

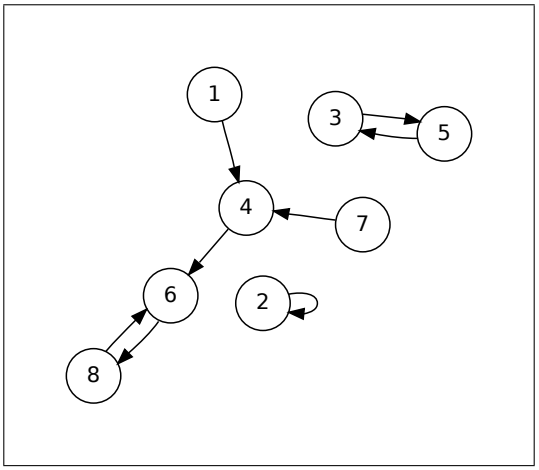


Figure 29: **Linear notation for a finite transformation:** $([[1,7;4];6],8)(3,5)$**. With this notation it is easy to see whether the transformation has a nontrivial cycle or not, one has to look for the round brackets. The traditional notation would be** $\left(\begin{smallmatrix}1&2&3&4&5&6&7&8\\4&2&5&6&3&8&4&6\end{smallmatrix}\right)$**.**

where target is the point in the cycle. If a source point also has incoming edges from other points the same square bracket structure is applied again recursively. We can say that the points in the cycle are sinks of trees. Parentheses indicate the existence of a nontrivial permutation of the sink elements of the trees, but not of their sources:

$$\big([\text{ sources}_1;\ \text{target}_1\ ],\ldots,[\text{ sources}_k;\ \text{target}_k\ ]\big).$$

This corresponds to the cycle

$$(\text{ target}_1,\ldots,\ \text{target}_k)$$

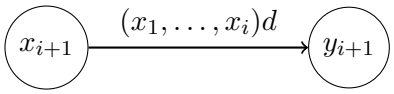
but at the same time it contains information on transient states. The order is arbitrary if there is more than one component. Our notation is slightly different to [38] as we do not use square brackets for a singleton source. Moreover, for constant mappings we just use $[\to n]$, where n is the image. This modified notation is used in `SgpDec`. Here are some examples (for an example of a functional digraph see Figure 29):

- The cyclic notation is retained for permutations $\left(\begin{smallmatrix}1&2&3\\1&2&3\end{smallmatrix}\right)=()$, $\left(\begin{smallmatrix}1&2&3\\2&3&1\end{smallmatrix}\right)=(1,2,3)$
- An elementary collapsing looks like this: $\left(\begin{smallmatrix}1&2&3\\3&2&3\end{smallmatrix}\right)=[1;3]$
- A 'line': $2\mapsto 3\mapsto 1$ $\left(\begin{smallmatrix}1&2&3\\1&3&1\end{smallmatrix}\right)=[[2;3];1]$
- A transposition with a third point collapsing into the cycle: $\left(\begin{smallmatrix}1&2&3\\2&1&2\end{smallmatrix}\right)=(1,[3;2])$
- A constant: $\left(\begin{smallmatrix}1&2&3\\1&1&1\end{smallmatrix}\right)=[\to 1]$.

4.3.3 Cascaded automata and abstract number systems

Integrating dependency functions arguments into the labels Cascaded operations can be visualized in a tree format, where the paths are labelled by states from the corresponding level and the nodes contain the local action (the value of the dependency function) [75]. However, this view displays only one element of the cascaded structure and thus a component of one particular level is scattered in several different diagrams (the number of generators). Therefore we need a diagram that shows the components as automata and still retains information of

the dependency functions. The idea is that we use a composite label for the state transitions: we augment the label of the cascaded operation with a coordinate tuple as the argument of its dependency function on that particular level. So $(x_1, \ldots, x_i)a$ means that the cascaded operation a acts on component $i+1$ by the state transition that it labels. The basic idea was used already in the late 60s but the state vector appeared explicitly in the label of a state transition first in [64] then it was used in [5]. For instance, in a cascaded structure, given the current states on the first i levels are $(x_1, \ldots, x_i)$, $x_k \in X_k$ and we have a dependency function d giving the value $s_{i+1} = d(x_1, \ldots, x_i)$, then for two states x_{i+1} and y_{i+1} of the $i+1$th level we will have a labelled arrow



supposing that $x_{i+1} \cdot s_{i+1} = y_{i+1}$.

This concept can probably be explained more easily through the example of a simple binary counter. Figure 30 provides such an example, but it is actually trying to achieve 4 things simultaneously:

- Provide an easily understandable example of what we mean by 'coordinatisation'.
- Provide an example of the tiling notation (that will be discussed next) through a visualisation of the decomposition of the cyclic group of 16 elements, C_{16}. Although the holonomy decomposition cannot be applied to a group and, therefore, such a tiling picture is not applicable, we are showing it anyway because it helps fix a couple of useful concepts that generalise to semigroup decompositions.
- Demonstrate the dependency functions through the familiar concept of the carry bit.
- Prepare the ground, therefore, for the introduction of the abstract concept of a finite-state automaton as defining a 'customised' generalised number system (to be discussed below).

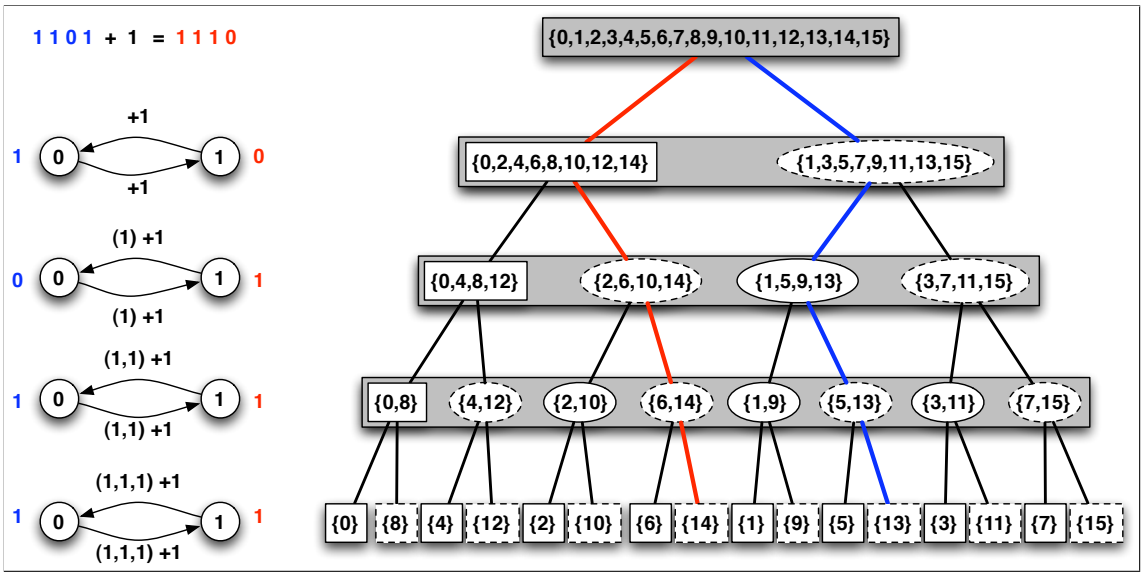


Figure 30: **Coordinatisation of the numbers 13 and 14 expressed as two elements of** C_{16}

Since this figure is trying to accomplish so many things, it will take a little time to unpack. We start with the familiar binary addition of 13 + 1 = 14, which is shown on the left of the figure. Each position in a positional number system can be seen as a different 'coordinate'.

Thus, following the concept of Krohn-Rhodes decomposition, we can arrange each coordinate as a different level in the decomposition. The least significant bit (LSB) is at the top, the most significant bit (MSB) at the bottom. What we have done is to depict each level as a 'mini-automaton', where the 'states' are the possible values of the binary digit, and the state transition label is none other than addition by a +1 input. But we notice that the labels look more complicated than that, so something is afoot.

Proceeding, 13 is shown as the binary number 1101 as a list of blue 1s and 0s arranged vertically on the far left. If we now add 1 to 13, we follow the usual rules and flip the LSB. Whether the higher bits (at the lower levels in this picture) are flipped or not *depends* on the levels above. For example, the second level will flip if the top level is a 1. This is shown as a '(1) +1', which means, 'If the state of the level above is a 1, then add 1, i.e. flip the state at the current level. Similarly, the third level will flip only if *both* the levels above are 1s, otherwise it will stay whatever it is. This shows what we meant when we said that the choice of semigroup element at a given level (+1 or identity, i.e. +0) depends on the states above. This carry bit rule of course extends to as many levels as there are digits representing the number in binary.

Finally, we describe the tiling notation. At each level a long gray rectangle indicates the presence of one or more permutation groups in one or more of the levels below. Because we are decomposing a group, clearly we have groups at each level, hence each level only has rectangles. Within the rectangles we can recognise the presence of a subgroup (the collection of numbers with 0, since 0 is the identity) or coset representative and of its coset(s). The subgroup is indicated with another rectangle, whereas its cosets are shown as ovals.

The number systems we use arise from nothing other than the choice of coordinatisation.[25] In the case of the binary system, each level is divided into two subsets, which we have indicated with a solid and a dotted line, respectively. In fact, we can verify that in switching from the blue line to the red line the binary digits that flip from 1 to 0 or vice versa correspond to the the 'coordinates' of 13 switching from the dotted to the solid subsets, or vice versa. Where such a switch does not occur, the digit remains the same. With this preparatory conceptual work, we can now look at a more abstract example.

Figure 31 shows a randomly generated automaton of 5 states and an input alphabet of 2 symbols, x and y. Although this is quite a simple automaton, it has plenty of complexity and a real semigroup of transformations.

The tiling picture shows that there is a permutation group associated with the 3 states $\{3, 4, 5\}$, since it has gray background. This group turns out to be the flipping of state 4 with state 5, i.e. it is a C_2, since the transformation of 4 to 5 is clearly reversible under the input y. The dotted lines indicate states that can never be returned to once the system leaves them (appropriately dubbed 'Garden of Eden states').

On the left, we see the sub-automata at each level of the decomposition. The limitations of using a notation meant to depict the holonomy decomposition of semigroups to decompose a group, in the previous example, now becomes clearer, since each sub-automaton actually sits between levels of the tiling picture rather than at the same level. In any case, in this figure the gray arrows indicate irreversible transformations, whereas the solid black arrows indicate group

[25] There are interesting examples of mixed-radix number systems that have arisen in different cultures seemingly as a reflection of different understandings of structure. Mixed-radix systems don't have the same set of digits at each level, like e.g. the Mayan number system which is positional but uses a mixture of base 20 and 18, or the ancient near Eastern base 60 systems, which to represent a base 60 digit (0-59) decompose it into units (0-9) and tens (0-5).

elements, as provided by SgpDec. However, this example is simple enough that it can be done by hand. In particular, the states of the sub-automaton at the top level is formed simply by the image sets of the original state set $\{1, 2, 3, 4, 5\}$ under the action of the two inputs. This is shown in Figure 32. Finally, the bottom level of Figure 31 shows how the state transitions depend on the system being in specific states in the level above.

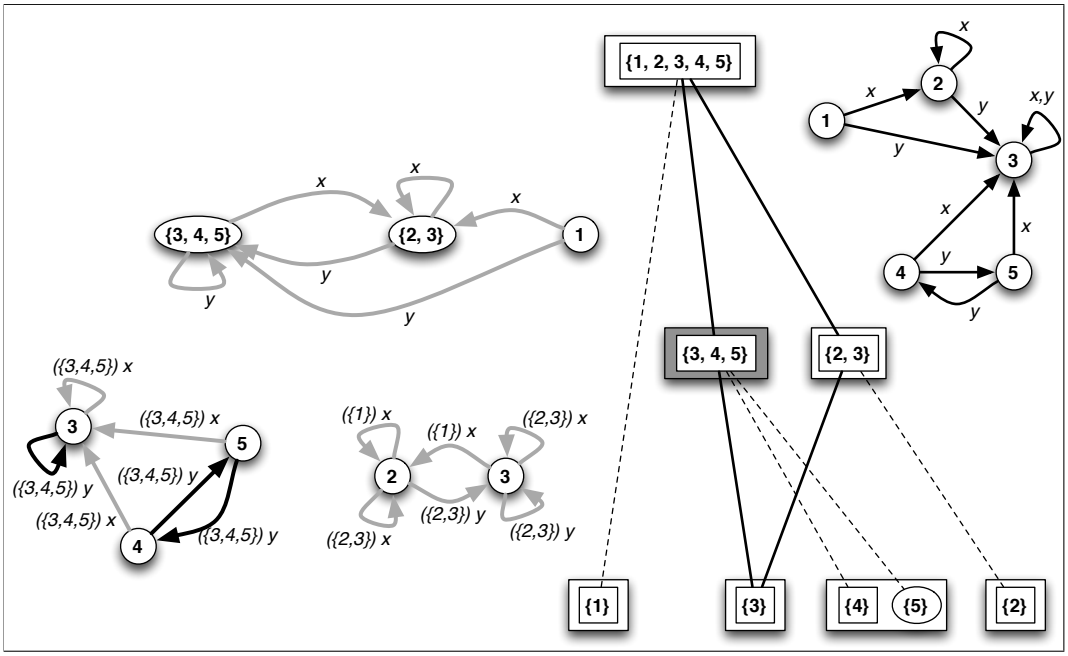


Figure 31: **Graphical notation for a randomly generated automaton (top right), the skeleton of its holonomy decomposition (tiling picture, middle) and the components of the hierarchical levels with the dependency functions (left).**

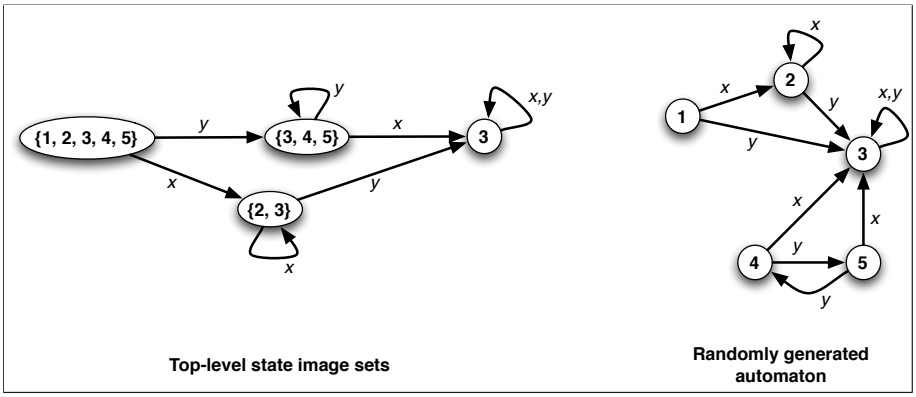


Figure 32: **Obtaining the image sets for the top level from the original automaton**

Taking a step back, we can see how we have treated this automaton in a way that is entirely analogous to a number, although in a rather more abstract way. The analogue of adding +1 to a binary number becomes the provision of an input symbol to the automaton. What made the number example of Figure 30 particularly simple is that the *inputs* and the *states at each level* all came from the same binary alphabet. For a general automaton this is clearly not going to be the case very often, if ever. Furthermore, at each level in a number expressed in a positional number system we have *one* mini-automaton. In this automaton example, on the other hand, at the bottom level two sub-automata are operating, each with its own state transitions; therefore, the bottom level is characterised by *two* states in parallel, which is a rather confusing way to think of a number. Finally, each state of the automaton corresponds to a different number

in the number system that the automaton defines. As a consequence, we reach the surprising conclusion that from this abstract perspective there is no difference between 'computing' and 'counting'.

Hopefully the power of the coordinatisation perspective to uncover the inner structure of automata is becoming clear, in the sense that a lot more information is provided by these diagrams relative to what one can evince from the original automaton. In any case, we notice an interesting parallel between this concept of an automaton as a generalised number system and the same idea applied to dynamical systems, as discussed in Section 2.1.2.

One arrow for constant maps The components of the holonomy decompositions are permutation-reset semigroups (groups together with constant maps for all states), therefore arrows for constant maps will be abundant in the diagram of the component automaton. Since we know that from all states there is always an arrow with the label of the constant map to target state (the image of the constant map), there is no need to draw them explicitly. Instead, a short wide arrow can indicate the existence of a constant map. One objection may be that in the case of a pure reset automaton the state transition diagram becomes disconnected, but actually this is desirable as the nonexistence of a group component is immediately clear.

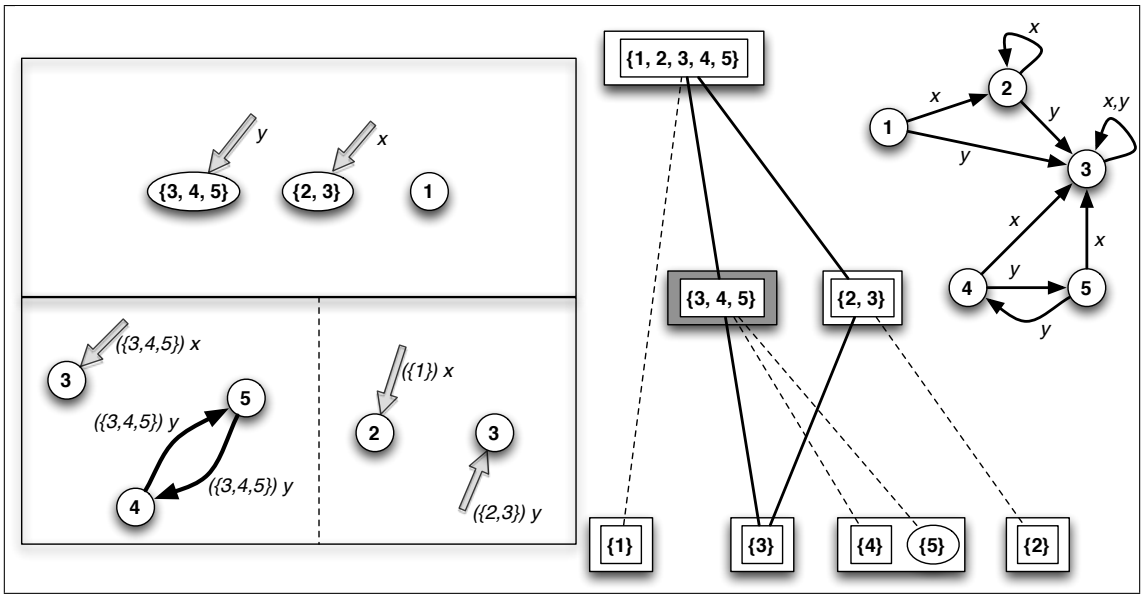


Figure 33: **Advanced graphical notation for the same randomly generated automaton. Thick arrows replace the arrows for a constant. The effect that the component automaton splits up into disconnected pieces is desirable as it shows that the states are not connected by a symmetry operation.**

In preparation for the more complicated examples discussed below, we show a very compact and low-resolution way to represent the semigroup decomposition of the random automaton. Figure 34 shows that this automaton is decomposable into just two levels. The 1 at the top level means that a trivial group is acting on the whole set of states $\{1, 2, 3, 4, 5\}$ to generate the first level of image sets. 'Trivial' here means that nothing comes back, both x and y give rise to irreversible transformations. The second level shows the presence of another trivial group, i.e. the irreversible transitions into the subset of states $\{2, 3\}$, and a cyclic group of order 2. C_2 refers to the permutation of states 4 and 5 under the action of the y input.

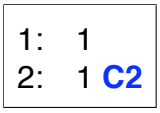


Figure 34: **Compact notation for showing the decomposition of the random automaton**

4.4 Technical advances

4.4.1 History of Computer Implementations

No matter how sound the underlying theory, developing useful applications is impossible if we do not have the practical tools to actually apply the theory. In the case of algebraic automata theory the practical tool is a computational implementation of the holonomic decomposition of arbitrary transformation semigroups. Unfortunately for the Krohn-Rhodes theory, there was no available computational tool for forty years. Therefore, except for special classes of automata where the decomposition can be established theoretically (e.g. full transformation semigroups, monogenic semigroups), some tedious hand calculations [81], and a few small example exercises [33], there was no possibility to study bigger examples of decompositions.

The first computational implementations were achieved at the University of Hertfordshire [27] and already surpassed the capabilities of pen and paper calculations, but they did not allow interactive calculations and were difficult to use by non-experts. Therefore a decision was made to include the semigroup decomposition package into the GAP computational algebra platform [39], the most advanced and open-source computer algebra system for computational group theory. The development started in 2006 but until the UH development team joined the OPAALS project only the Lagrange decomposition [31] (for permutation groups) had been completed for the software package. The new challenges in the OPAALS project required further advancements in the software development and eventually led to a mature software package [32].

Currently, `SgpDec`[32] is capable of calculating decompositions of semigroups of size tens of thousands (acting on around 50 states). This is beyond doubt a significant step towards realizing the vision of 'engineering' dynamical systems.

4.4.2 Interactive computing for experimental exploration

Since computational implementations for algebraic decompositions are still experimental we put special emphasis on the interactive capabilities of the `SgpDec` package [32]. The calculations can be done in smaller pieces and can be investigated or 'splashed', i.e. the automatically generated figures can be displayed immediately, supporting visualizations of the decompositions, cascaded operations, as well as other structures.

4.4.3 Improving scalability

There are different ways of doing hierarchical decompositions of semigroups, but we are interested in those dealing with transformation semigroups, as the state set can be considered as analogous to the phase space of a dynamical system. The holonomy decomposition works directly with the state set, therefore it is the primary choice for a computational implementation. The

decomposition itself is a complicated algorithm with several stages. We have improved scalability of the software in the following ways:

- Integrating functionalities from other `GAP` packages.
  - The `MONOID` [70] contains the efficient orbit calculation algorithm used for producing the skeleton of the holonomy decomposition.
  - The `orb` [74] package contains excellent hashtable functionality. Hashtables are used in many places when we keep track of a growing set of elements.
- Using minimizations for sequences of state transitions by cutting off redundant loops. These 'straight words' can be used for reducing memory footprint and processing time [25].

4.5 Experiments on the algebraic models of the p53-mdm2 system

For studying the p53-mdm2 system algebraically we start from a Petri net (Fig. 35). A discussion of this Petri net and of how it models the p53-mdm2 system can be found in [24, 89, 15]. From the Petri net we can derive an automaton by fixing the capacities of the places (for the exact method see [29]), but increasing the capacities of the places can lead to automata with so many states that they become impossible to handle due to scalability issues. The simplest case is when we only distinguish between the presence (above a certain threshold) and the absence of some particular substrate (capacity 2). In this case we get an automaton with 16 states (Fig. 36) with a corresponding semigroup that has 316,665 elements and admits a 17-level decomposition (Fig. 37). This size may be too small for the model, the semigroup contains no SNAGs (simple non-abelian groups). Fortunately, recent advances in the scalability of the `SgpDec` [32] package allows us to study decompositions of bigger automata (Fig. 38).

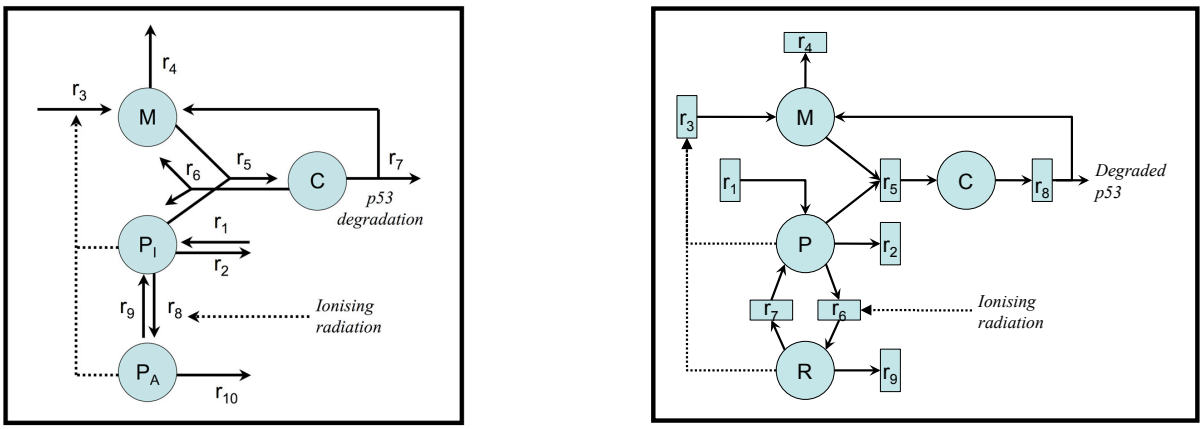


Figure 35: **(A) Schematic of the p53-mdm2 regulatory pathway [89], and (B) corresponding Petri net [24]. P = p53, M = mdm2, C = p53-mdm2, R = p53*.**

Table 2 summarises the salient features of the three examples we have examined.

It is clear that the next challenge is to find a way to reduce the number of levels, for example by compressing the information contained in the levels only with trivial group components.

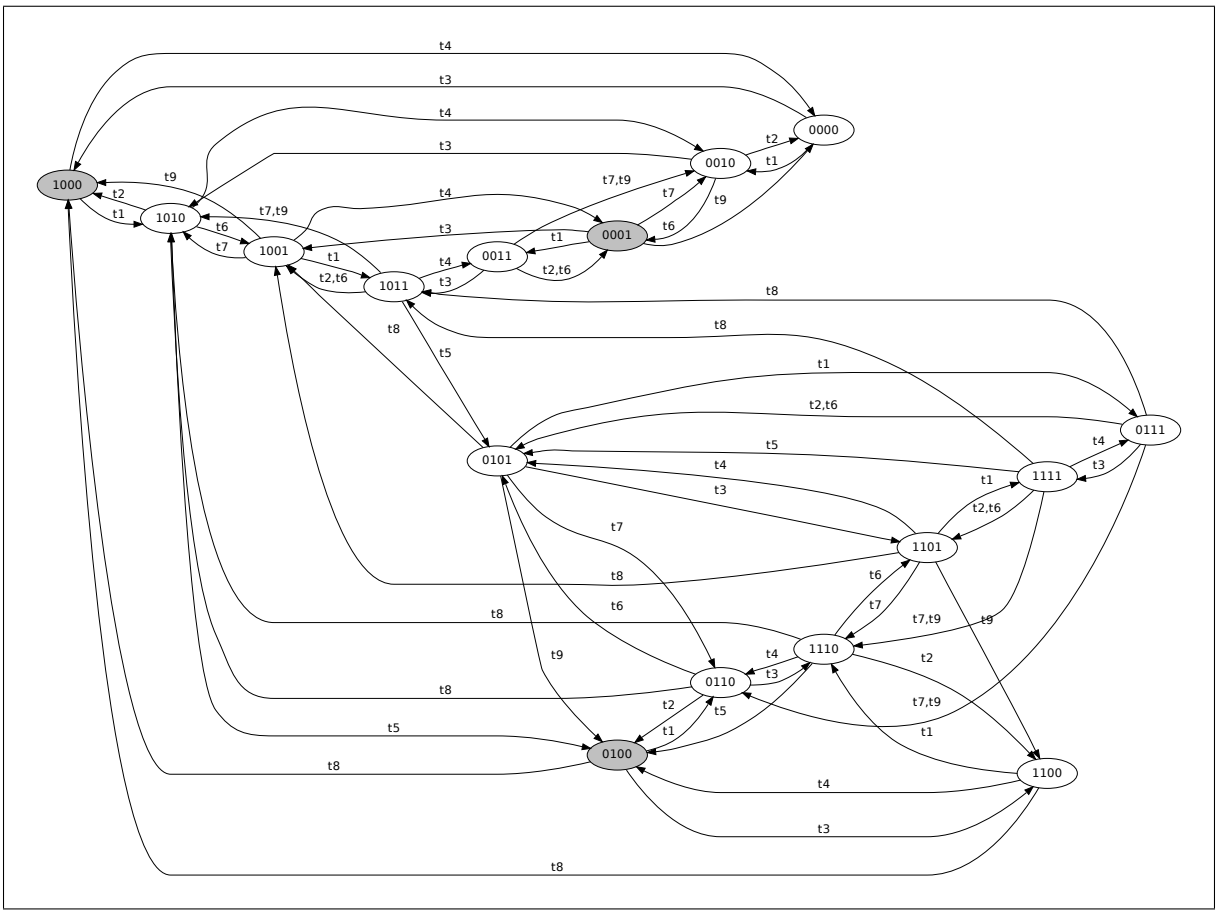


Figure 36: **Automaton derived from 2-level Petri net of the p53 system (16 states). The labels on the nodes encode the possible configurations for M, C, P and R (in this order). 0 denotes the absence (or presence below a threshold), 1 the presence (above the threshold) of the given type of molecule. For instance, 0101 means that C and R are present. The shaded states correspond to the state set** $\{3 = 1000, 5 = 0001, 8 = 0100\}$**.**

```
 1:  1
 2:  1
 3:  1
 4:  1
 5:  1 1
 6:  1 1
 7:  1 1 1
 8:  1 1 1 1
 9:  1 1 1 1
10:  1 1 1
11:  C2
12:  1
13:  1 1
14:  1 1 1
15:  S3
16:  1
17:  C2
```

Figure 37: **The components of the holonomy decomposition of the 16-state p53 automaton. Each line corresponds to a hierarchical level, several components on the same level indicate parallelism within one level. The top level is 1, 17 is the lowest level. Level 17 depends on changes at levels 1 to 16, whereas the top level's operation is independent of the states of lower levels. The components are permutation-resets (a group augmented with constant maps), but only the group components are indicated (the trivial group is denoted by 1).**

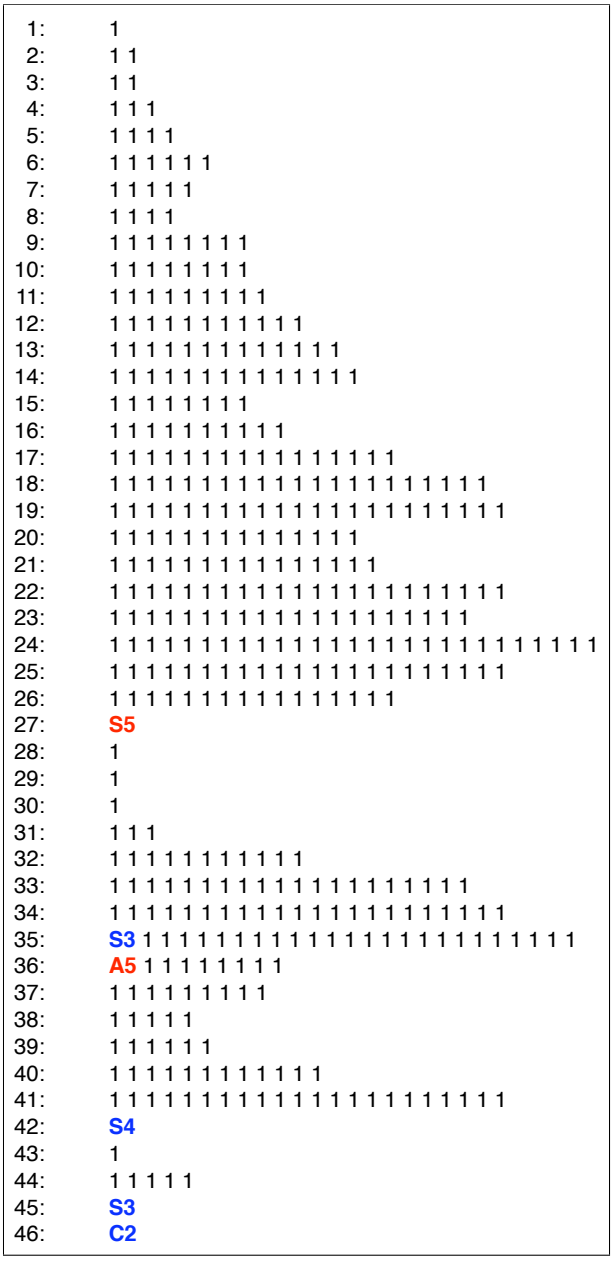


Figure 38: **Groups occurring at each level of the holonomy decomposition of a 36-state p53 automaton (having 3 different levels of concentration for M and C, whereas only two different concentration thresholds are distinguished for P and R). A SNAG, A_5, the alternating group on 5 points, does appear among the group components. Although S_5 is not a simple group, it is shown in red because it contains A_5.**

	Random Automaton	**16-State p53 System**	**36-State p53 System**
Number of states	5	16	36
Number of levels	2	17	46
Order of semigroup	4	316,665	Unknown, still being calculated by a Mac server with 12GB RAM, 260 hrs CPU time and counting...

Table 2: **Comparison of characteristic dimensions of the automata examined**

5 Conclusion

5.1 Highlights of general results in interaction computing research 2003-10

Within the interaction computing research agenda, which started in 2002 with the preparation of the DBE project proposal, the main 'discoveries' to date have been the following:[26]

- Krohn-Rhodes theory [81] and the work of UH, who subsequently joined the project.
- The possible interpretation of the permutation groups of state subsets embedded within the transformation semigroup of a given automaton as a mechanism for parallel, cyclic, and not necessarily in-phase computation ([16]: 23-24; [24]).
- The theoretical framework connecting automata structure and behaviour, and automata behaviour and specification languages through category theory ([21]: 120; [81]: 57), shown in Figure 39.
- The realisation that global Lie symmetries are probably too restrictive to be applicable to the analysis of metabolic pathways. As we mentioned in D1.3 [15] Lie groupoids are likely to be helpful here as they correspond to symmetries that are 'local' in some sense. For example, biological systems do not operate at all possible temperatures, but only over a fairly limited range of temperature. However, the type of mathematical thinking required is very different to what we have been working on in the past few years (with the exception of [12, 16, 51]). It seems best to focus on the discrete mathematics perspective of the present report, which is more likely to bear fruit in the short term. Any such results could then be helpful also in developing a Lie group/groupoid-based approach.
- The probable relevance of Ross Ashby's Law of Requisite Variety [2] to the architecture of self-organising systems. In other words, the self-organisation of metabolic systems and the self-organisation of evolutionary systems might be seen as instantiations of this same abstract law into these two very different contexts. In both cases we would therefore expect to find a common mathematical structure; for instance, a reliance of 'local' behaviour on 'global' context, where local and global are relative and recursive concepts in the sense that they apply to each pair of adjacent layers in a nested hierarchy. The difference is that in the case of evolution the 'nesting' takes place along the time dimension – and we call it 'history', or 'memory', or 'path-dependence' – whereas in the case of metabolic systems it takes place along an axis that measures system scale. Thus, from the point of view of the law of requisite variety metabolism could be seen as a 'projection' of evolution along the time axis and onto the 'plane' perpendicular to the time axis that defines 'now' (thereby resurrecting a strange form of recapitulation [18]). In both cases, in fact, we observe spontaneous order construction of unbelievable complexity, but on totally opposite time-scales. The fact that the dynamical self-organisation of metabolic and other real-time systems appears to depend on a non-linear mathematical structure could then be explained by the fact that non-linearity provides the coupling between different degrees of freedom that is necessary for different scales to be able to communicate (as in the well-known energy cascade of turbulent flows) – a role played by memory mechanisms in the evolutionary case. The possibility of a mathematical equivalence between temporal and scale-dependent phenomena at the architectural level would reinforce the hunch that algebraic automata theory (conceptual placeholder for machine structure) and symbolic dynamics (conceptual placeholder for machine behaviour and algorithms) can be related in deep and useful ways. This is echoed by the familiar relationship between data structures and algorithms in computer science and software engineering, a discussion that is developed further below.

5.2 Main outcomes and results of this report

- Discovery that the p53-mdm2 system exhibits homoclinic behaviour, and is therefore amenable to the same kind of analysis as the horseshoe map of chaos theory.

[26] We include the weaker discoveries of finding that a particular existing theory is important, as well as the 'negative' discoveries that a particular theory is *not* relevant.

- Discovery of two kinds of occurrences of two SNAGs (simple non-abelian groups) even in a relatively coarse discretisation of the p53-mdm2 system (Chapter 4).
- The two preceding points are both indicative of something significant within this particular dynamical system. This is not well-understood but symbolic dynamics (Chapter 2) might create a bridge between the two methodologies that could lead to more insights for interaction computing as it is capable of connecting dynamical systems to algebraic automata theory, network coding, and formal languages (see Figure 5).
- The concept of abstract number system as equally applicable to automata and to dynamical systems expressed through symbolic dynamics, and therefore possibly providing a formal relationship between the coordinatisation of an automaton and the coordinatisation of the languages it recognises. The potential benefit of this line of thinking is to reduce structural and behavioural/algorithmic aspects of computation to the same abstract form of 'arithmetic'. There is such a known relationship called the 'Straubing wreath product principle', relating languages to wreath product decompositions (plus other principles developed to some extent by Eilenberg and by others), but the relation to number systems or dynamical systems is not well known and not really clear since not the whole wreath product but only a cascaded substructure is what really matters. This question is probably worth at least one PhD and will be investigated in our next steps.
- Discovery of a new class of potential cancer drugs, the Tenovins.
- Completion of and significant improvements to the SgpDec transformation semigroup decomposition computational algebra GAP package.
- Notational advances for working with cascaded structures and in denoting wreath product decompositions in visualisable form.

5.3 Critical discussion and next steps

Figure 39 was arrived at after a 4-year collaboration with Daniel Shreckling of the Institute of IT-Security and Security Law, University of Passau, in the context of the BIONETS project. It represents a synthesis of three different points of view:

- A physics and dynamical systems view of metabolic systems
- A logic-based and formal view of engineering specification
- An algebraic bridge between the two that aims to inform computer science and the software engineering process

The vertical two-headed arrow in Figure 39 can be seen as embodying the essence of Engineering. Moving down means that given a structure we wish to calculate its behaviour. This is a relatively easy task and mathematically it generally corresponds to a well-posed single-valued problem. This is true for traditional engineering disciplines as well as for automata. This direction of calculation goes by the name of 'direct' problem and is associated with *engineering analysis*. By contrast, moving up along the same two-headed arrow is orders of magnitude more difficult. If we identify 'automata behaviour' with the concept of engineering requirements, then moving up the arrow is none other than *engineering design*, sometimes redundantly called *inverse* design. Mathematically, the design or 'inverse' problem is very seldom well-posed; generally it is multi-valued, meaning that different structures can give rise to the same behaviour.

Nature has solved the really hard inverse problem through evolution, thanks to its extremely slow and incremental trial-and-error process. Ontogeny, morphogenesis and gene expression can be seen as a 'meta' order-construction processes whose characteristic order-construction time

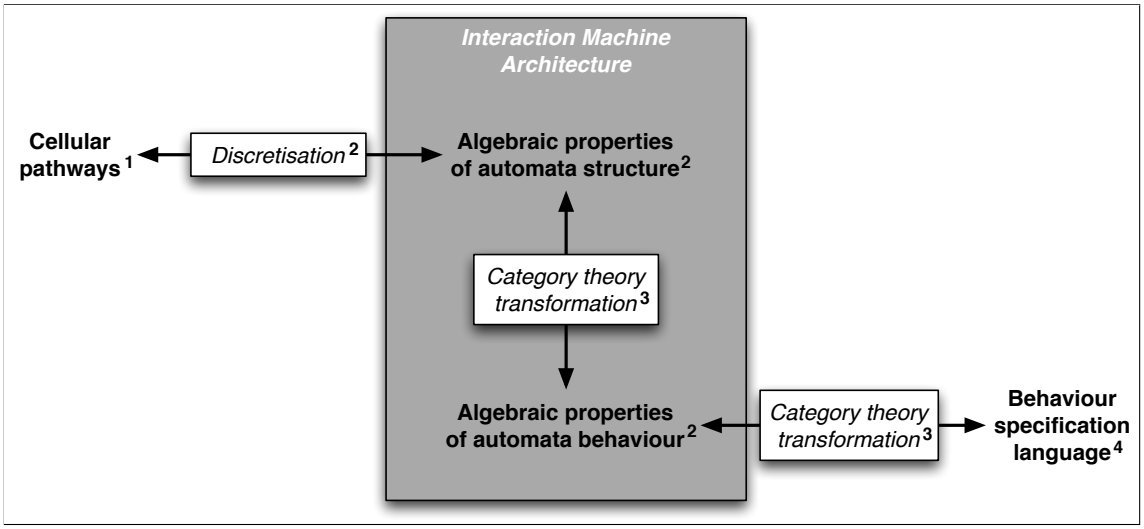


Figure 39: **Theoretical areas of relevance to interaction computing. Legend: 1 = [89]; 2 = [24]; 3 = [21, 22, 16, 56, 83, 19]; 4 = future projects/papers (e.g. [6])**

scales are approximately 10^6-10^8 times smaller (i.e. faster) than evolution. In order to work so quickly, clearly metabolic processes must benefit from the presence of suitable constraints that eliminate the multi-valuedness of the problem and select the 'viable' solution – where viability is defined relative to a suitable fitness function. Our argument all along has been that these constraints are algebraic in character. Our results to date support, but do not yet prove, this hypothesis.

The two category theory transformations shown in the figure actually have rather different character. The transformation between specification and automata behaviour (i.e. sequential machines that formalise mathematical functions), shown as the horizontal two-headed arrow at the bottom-right of the figure, is analogous to a translation between different human languages. Category theory enables tranformations that preserve the algebraic structure in the two cases.

On the other hand, the vertical transformation is properly called an adjunction [42] and is dicussed in greater depth in [21]. In this case category theory gives us a formal framework through which we can relate structure and behaviour, one of the ubiquitous concepts of human and biological experience. In biology it is better known as the relationship between structure and function, to which we added organisation in a previous report [15]. These same three 'usual suspects' are recognisable also in this figure and in this discussion. 'Organisation' is here shown as the architecture of the interaction machine, which is most probably an elaboration of the Law of Requisite Variety discussed above.

A further hypothesis that has gradually taken shape based on these ideas is that the mathematical relationship we postulated between data structure and dynamic (algorithmic) structure, above, could provide precisely the mathematical constraint necessary to render the inverse problem well-posed. This raises an interesting epistemological question. We notice that the familiar software engineering development process is *also* miraculously fast, relative to evolution, because like other engineering activities it relies on formalisms that are well-adapted to the human mind, i.e. programming languages. Therefore, it would seem that through the cognitive activity of software engineering we are able to produce computational structures that might indeed approximate a good level of compatibility between the structural and dynamic aspects of systems. Thus, an interesting research question is to ask whether by formalising the interdependence between them we might not be able to speed up and automate the software engineering development process to levels that today are unthinkable.

A simple example to explain the last point above might be helpful. Imagine a primitive society trying to build a civil engineering structure such as a bridge or a dome in such a way that it is as light as possible. Assume further that for aesthetic reasons the structure needs to be built with the same material throughout. Trial and error combined with ingenuity and intuition would eventually lead to a constant-stress structure, i.e. obtained by a bridge or wall thickness that thins out as one moves up along the structure. Beautiful and fully functional structures could be built in this way, and 'rules of thumb' could be developed, and passed on between generations of builders. There is, technically, no need to develop a mathematical theory to prove that the thickness distribution of such constant-stress structures happens to be parabolic. But once some mathematician were to realise that this is the case, the building processes might become more self-assured, and the builders might attempt more ambitious buildings, such as the Pantheon in Rome, which is an example of precisely such a structure. Thus, the point of our research in interaction computing is to uncover hidden and yet unknown mathematical structures, some of which are likely to come from biology – but that might also be already present in the software artefacts we have been building for decades!

There are two areas of current research in computer science that seem compatible with our mathematical perspective: Egon Börger's Abstract State Machines (ASM) [6] and Robin Milner's bigraphs [69]. As an outgrowth of the π-Calculus, Milner's formalism is most suitable for modelling the behaviour *between* processes, i.e. of the network and including mobility. The ASM approach, on the other hand, provides a very detailed formalisation of the software engineering process for creating *single* applications, which we can assume to be running locally. The appeal of the ASM approach is that it elevates programming language constructs to the status of finite-state machines, whereas the appeal of the bigraphs is that they are built on a category theory foundation. Therefore, we see the possibility to integrate these two formalisms in order to cover both the 'local' (ASM) and the 'global' (bigraphs) scales, utilising our algebraic perspective as a scale-invariant unifying framework for the specification of distributed environments capable of supporting self-organising behaviour through interactions and communications between ASMs.

References

[1] E Altman, P Dini, D Miorandi, and D Schreckling, editors. *Paradigms for Biologically-Inspired Autonomic Networks and Services: The BIONETS Project eBook*. BIONETS, 2010. Available from: `http://www.bionets.eu`.

[2] William Ross Ashby. *An introduction to cybernetics*. Chapman & Hall Ltd., London, 1956. Available from: `http://pespmc1.vub.ac.be/books/IntroCyb.pdf`.

[3] Gonca Ayik, Hayrullah Ayik, and John M. Howie. On factorisations and generators in transformation semigroups. *Semigroup Forum*, 70:225–237, 2005.

[4] E Batchelor, C S Mock, I Bhan, A Loewer, and G Lahav. Recurrent initiation: A mechanism for triggering p53 pulses in response to dna damage. *Mol Cell*, 30:277–289, 2008.

[5] Anne Bergeron and Sylvie Hamel. From cascade decompositions to bit-vector algorithms. *Theoretical Computer Science*, 313:3–16, 2004.

[6] E Börger and R Stärk. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer-Verlag, 2003.

[7] G Briscoe, P Dini, Nicolas Oros, A Yinusa, M Buck, and T V Prabhakar. *D1.5-Integration of gene expression computing and operational closure in the software ecosystem architecture*. OPAALS Project, 2010. Available from: `http://files.opaals.eu/OPAALS/Year_4_Deliverables`.

[8] C J Brown, S Lain, C S Verma, A R Fersht, and D P Lane. Awakening guardian angels: drugging the p53 pathway. *Nat Rev Cancer*, 9:862–873, 2009.

[9] A Ciliberto, B Novak, and J J Tyson. Steady states and oscillations in the p53/mdm2 network. *Cell Cycle*, 4:488–493, 2005.

[10] A R Clarke and M Hollstein. Mouse models with modified p53 sequences to study cancer and ageing. *Cell Death Diff*, 10:443–450, 2003.

[11] L N De Castro and J Timmis. *Artificial Immune Systems: A New Computational Intelligence Approach*. Springer, London, 1 edition, November 2002.

[12] P Dini. *D18.4-Report on self-organisation from a dynamical systems and computer science viewpoint*. DBE Project, 2007. Available from: `http://files.opaals.eu/DBE`.

[13] P Dini and E Berdou. *D18.1-Report on DBE-Specific Use Cases*. DBE Project, 2004. Available from: `http://files.opaals.eu/DBE`.

[14] P Dini, G Briscoe, A J Munro, and S Lain. *D1.1: Towards a Biological and Mathematical Framework for Interaction Computing*. OPAALS Deliverable, European Commission, 2008. Available from: `http://files.opaals.eu/OPAALS/Year_2_Deliverables/WP01/`.

[15] P Dini, G Briscoe, I Van Leeuwen, A J Munro, and S Lain. *D1.3: Biological Design Patterns of Autopoietic Behaviour in Digital Ecosystems*. OPAALS Deliverable, European Commission, 2009. Available from: `http://files.opaals.eu/OPAALS/Year_3_Deliverables/WP01/`.

[16] P Dini, G Horváth, D Schreckling, and H Pfeffer. *D2.2.9: Mathematical Framework for Interaction Computing with Applications to Security and Service Choreography*. BIONETS Deliverable, European Commission, 2009. Available from: `http://www.bionets.eu`.

[17] P Dini, M Iqani, R Rivera-León, A Passani, S Moschoyiannis, O Nykänen, D Pattanaik, and J Chatterjee. *D12.10: Foundations of the Theory of Associative Autopoietic Digital Ecosystems: Part 3*. OPAALS Deliverable, European Commission, 2009. Available from: `http://files.opaals.eu/OPAALS/Year_3_Deliverables/WP12/`.

[18] P Dini, A J Munro, M Iqani, F Zeller, S Moschoyiannis, J Gabaldon, and O Nykanen. *D1.2: Foundations of the Theory of Associative Autopoietic Digital Ecosystems: Part 1*. OPAALS Deliverable, European Commission, 2008. Available from: `http://files.opaals.eu/OPAALS/Year_2_Deliverables/WP01/`.

[19] P Dini and D Schreckling. A Research Framework for Interaction Computing. In Fernando A B Colugnati, Lia C R Lopes, and Saulo F A Barretto, editors, *Digital Ecosystems: Proceedings of the 3rd International Conference, OPAALS 2010*, pages 224–244, Aracaju, Sergipe, Brazil, 22-23 March, 2010. Springer LNICST.

[20] P Dini and D Schreckling. On Abstract Algebra and Logic: Towards their Application to Cell Biology and Security. In E Altman, P Dini, D Miorandi, and D Schreckling, editors, *Paradigms for Biologically-Inspired Autonomic Networks and Services: The BIONETS Project eBook*, 2010. Available from: `http://www.bionets.eu`.

[21] P Dini, D Schreckling, and G Horváth. Algebraic and Categorical Framework for Interaction Computing and Symbiotic Security. In E Altman, P Dini, D Miorandi, and D Schreckling, editors, *Paradigms for Biologically-Inspired Autonomic Networks and Services: The BIONETS Project eBook*, 2010. Available from: `http://www.bionets.eu`.

[22] P Dini, D Schreckling, and L Yamamoto. *D2.2.4: Evolution and Gene Expression in BIONETS: A Mathematical and Experimental Framework*. BIONETS Deliverable, European Commission, 2008. Available from: `http://www.bionets.eu`.

[23] Pál Dömösi and Chrystopher L. Nehaniv. *Algebraic Theory of Finite Automata Networks: An Introduction*, volume 11. SIAM Series on Discrete Mathematics and Applications, 2005.

[24] A Egri-Nagy, P Dini, C L Nehaniv, and M J Schilstra. Transformation Semigroups as Constructive Dynamical Spaces. In Fernando A B Colugnati, Lia C R Lopes, and Saulo F A Barretto, editors, *Digital Ecosystems: Proceedings of the 3rd International Conference, OPAALS 2010*, pages 245–265, Aracaju, Sergipe, Brazil, 22-23 March, 2010. Springer LNICST.

[25] A. Egri-Nagy and C. L. Nehaniv. On straight words and minimal permutators in finite transformation semigroups. *LNCS Lecture Notes in Computer Science*, 2010. accepted.

[26] A. Egri-Nagy, C. L. Nehaniv, J. L. Rhodes, and M. J. Schilstra. Automatic analysis of computation in biochemical reactions. *BioSystems*, 94(1-2):126–134, 2008.

[27] Attila Egri-Nagy and Chrystopher L. Nehaniv. Algebraic hierarchical decomposition of finite state automata: Comparison of implementations for Krohn-Rhodes Theory. In *Conference on Implementations and Applications of Automata CIAA 2004*, volume 3317 of *Springer Lecture Notes in Computer Science*, pages 315–316, 2004.

[28] Attila Egri-Nagy and Chrystopher L. Nehaniv. Cycle structure in automata and the holonomy decomposition. *Acta Cybernetica*, 17:199–211, 2005. [ISSN: 0324-721X].

[29] Attila Egri-Nagy and Chrystopher L. Nehaniv. Algebraic properties of automata associated to Petri nets and applications to computation in biological systems. *BioSystems*, 94(1-2):135–144, 2008.

[30] Attila Egri-Nagy and Chrystopher L. Nehaniv. Hierarchical coordinate systems for understanding complexity and its evolution with applications to genetic regulatory networks. *Artificial Life*, 14(3):299–312, 2008. (Special Issue on the Evolution of Complexity),.

[31] Attila Egri-Nagy and Chrystopher L. Nehaniv. Subgroup chains and Lagrange coordinatizations of finite permutation groups. arXiv:0911.5433v1 [math.GR], 2009.

[32] Attila Egri-Nagy and Chrystopher L. Nehaniv. *`SgpDec` – software package for hierarchical coordinatization of groups and semigroups, implemented in the `GAP` computer algebra system, Version 0.5.19*, 2010. `http://sgpdec.sf.net`.

[33] Samuel Eilenberg. *Automata, Languages and Machines*, volume B. Academic Press, 1976.

[34] G Farmer, J Bargonetti, H Zhu, P Friedman, R Prywes, and C Prives. Wild-type p53 activates transcription in vivo. *Nature*, 358:83–84, 1992.

[35] S F Florman. *The Existential Pleasures of Engineering*. St Martin's Press, New York, 1976.

[36] C Frougny. Number Representation and Finite Automata. In F Blanchard, A Maas, and A Nogueira, editors, *Topics in Symbolic Dynamics and Applications*, volume 279 of *London Mathematical Society Lecture Notes Series*, pages 207–228. Cambridge University Press, 2000.

[37] T Fujiwara, D W Cai, R N Georges, T Mukhopadhyay, E A Grimm, and J A Roth. Therapeutic effect of a retroviral wild-type p53 expression vector in an orthotopic lung cancer model. *J Natl Cancer Inst*, 86:1458–1462, 1994.

[38] Olexandr Ganyushkin and Volodymyr Mazorchuk. *Classical Transformation Semigroups*. Algebra and Applications. Springer, 2009.

[39] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.4*, 2006. http://www.gap-system.org.

[40] Judith L. Gersting. *Mathematical Structures for Computer Science*. W. H. Freeman, 1982.

[41] Abraham Ginzburg. *Algebraic Theory of Automata*. Academic Press, 1968.

[42] Joseph A. Goguen. Realization is universal. *Mathematical Systems Theory*, 6(4):359–374, 1972.

[43] K E Gordon, I M M van Leeuwen, S Lain, and M A J Chaplain. Spatio-temporal modelling of the p53-mdm2 oscillatory system. *Math Model Nat Phenom*, 2009. in press.

[44] J Hadamard. Les surfaces a courbures opposés et leurs lignes geodesiques. *Journal de Mathematiques Pures et Appliqué*, 4:27–73, 1898.

[45] M Harvey, H Vogel, D Morris, A Bradley, A Bernstein, and LA Donehower. A mutant p53 transgene accelerates tumour development in heterozygous but no nullzygous p53-deficient mice. *Nat Cancer Rev*, 9:68–76, 1995.

[46] Y Haupt, R Maya, A Kazaz, and M Oren. Mdm2 promotes the rapid degradation of p53. *Nature*, 387:296–299, 1997.

[47] M W Hirsch, S Smale, and R L Devaney. *Differential Equations, Dynamical Systems, and an Introduction to Chaos*. Elsevier, Amsterdam, 2004.

[48] W. M. L. Holcombe. *Algebraic Automata Theory*. Cambridge University Press, 1982.

[49] M Hollis. *The Philosophy of Social Science: An Introduction, 2nd Ed*. Cambridge, 1994.

[50] G Horváth. *Functions and Polynomials over Finite Groups from the Computational Perspective*. The University of Hertfordshire, PhD Dissertation, 2008.

[51] G Horváth and P Dini. Lie Group Analysis of p53-mdm3 Pathway. In Fernando A B Colugnati, Lia C R Lopes, and Saulo F A Barretto, editors, *Digital Ecosystems: Proceedings of the 3rd International Conference, OPAALS 2010*, pages 285–304, Aracaju, Sergipe, Brazil, 22-23 March, 2010. Springer LNICST.

[52] S Kauffman. *The Origins of Order: Self-Organisation and Selection in Evolution*. Oxford University Press, Oxford, 1993.

[53] K Krohn and J Rhodes. Algebraic Theory of Machines. I. Prime Decomposition Theorem for Finite Semigroups and Machines. *Transactions of the American Mathematical Society*, 116:450–464, 1965.

[54] J P Kruse and W Gu. Modes of p53 regulation. *Cell*, 137:609–622, 2009.

[55] G Lahav, N Rosenfield, A Sigal, N Geva-Zatorsky, A J Levine, M B Elowitz, and U Alon. Dynamics of the p53-mdm2 feedback loop in individual cells. *Nat Gen*, 36:147–150, 2004.

[56] J Lahti, J Huusko, D Miorandi, L Bassbouss, H Pfeffer, P Dini, G Horváth, S Elaluf-Calderwood, D Schreckling, and L Yamamoto. *D3.2.7: Autonomic Services within the BIONETS SerWorks Architecture*. BIONETS Deliverable, European Commission, 2009. Available from: `http://www.bionets.eu`.

[57] S Lain, J J Hollick, J Campbell, O D Staples, M Higgins, M Aoubala, A McCarthy, V Appleyard, K E Murray, L Baker, A Thompson, J Mathers, S J Holland, M J R Stark, G Pass, J Woods, D P Lane, and N J Westwood. Discovery, *in Vivo* activity, and mechanism of action of a small-molecule p53 activator. *Cancer Cell*, 13:454–463, 2008.

[58] D P Lane. p53, guardian of the genome. *Nature*, 358:15–16, 1992.

[59] DP Lane. p53 from pathway to therapy. *Carcinogenesis*, 25:1077–1081, 2004.

[60] A Lavinguer, V Maltby, D Mock, J Rossant, T Pawson, and A Bernstein. High incidence of lung, bone, and lymphoid tumours in transgenic mice overexpressing mutant alleles of the p53 protooncogene. *Mol Cell Biol*, 9:3982–3991, 1989.

[61] R Lev Bar-Or, R Maya, L A Segel, U Alon, A J Levine, and M Oren. Generation of oscillations by the p53-mdm2 feedback loop: A theoretical and experimental study. *Proc Natl Acad Sci USA*, 97:11250–11255, 2000.

[62] D Lind and B Marcus. *An Introduction to Symbolic Dynamics and Coding*. Cambridge University Press, 1995.

[63] J Luo, M Li, Y Tang, M Laszkowska, R G Roeder, and W Gu. Acetylation of p53 augments its site-specific dna binding both *in Vitro* and *in Vivo*. *Proc Natl Acad Sci USA*, 101:2259–2264, 2004.

[64] Oded Maler and Amir Pnueli. On the cascade decomposition of automata, its complexity and its application to logic. DRAFT, 1994. November 15.

[65] H Maturana and F Varela. *Autopoiesis and Cognition. the Realization of the Living*. D. Reidel Publishing Company, Boston, 1980.

[66] H Maturana and F Varela. *The Tree of Knowledge. The Biological Roots of Human Understanding*. Shambhala, Boston and London, 1998.

[67] A R McCarthy, J J Hollick, and N J Westwood. The discovery of nongenotoxic activators of p53: building on a cell-based high-throughput screen. *Sem Cancer Biol*, 20:40–45, 2010.

[68] S M Mendrysa and M E Perry. The p53 tumor suppressor protein does not regulate expression of is own inhibitor, *MDM2*, except under conditions of stress. *Mol Cell Biol*, 20:2023–2030, 2000.

[69] Robin Milner. *The Space and Motion of Communicating Agents*. Cambridge, 2009.

[70] James Mitchell. *MONOID Version 3.1.4*, 2010. `http://turnbull.mcs.st-and.ac.uk/~jamesm/monoid/`.

[71] J Momand, G P Zambetti, D C Olson, D George, and A J Levine. The mdm2 oncogene product forms a complex with the p53 protein and inhibits p53-mediated transactivation. *Cell*, 69:1237–1245, 1992.

[72] N A M Monk. Oscillatory expression of hes1, p53, and nf-kappab driven by transcriptional time delays. *Curr Biol*, 13(16):1409–1413, 2003.

[73] M Morse and G A Hedliund. Symbolic Dynamics. *American Journal of Mathematics*, 60(4):815–866, 1938.

[74] Jürgen Müller, Max Neunhöffer, and Felix Noeske. *orb Version 3.6*, 2010. `http://www-groups.mcs.st-and.ac.uk/~neunhoef/Computer/Software/Gap/orb.html`.

[75] C L Nehaniv. *Global Sequential Coordinates on Semigroups, Automata, and Infinite Groups*. University Microfilms International, Ann Arbor, Michigan, USA, 1992. Ph.D. Thesis in Mathematics, University of California, Berkeley.

[76] C L Nehaniv. Algebraic Engineering of Understanding: Global Hierarchical Coordinates on Computation for the Manipulation of Data, Knowledge, and Process. In *Proceedings of the 18th Annual International Computer Software and Applications Conference (COMPSAC94)*, pages 418–425, Taipei, Taiwan, 1994. IEEE Computer Society Press.

[77] C L Nehaniv and J L Rhodes. The Evolution and Understanding of Hierarchical Complexity in Biology from an Algebraic Perspective. *Artificial Life*, 6:45–67, 2000.

[78] M Oren. Decision making by p53: life, death and cancer. *Cell Death Diff*, 10:431–442, 2003.

[79] K Popper. *The Logic of Scientific Discovery*. Routledge, London, 2002. reprinted translation of 1935 original Logik der Forschung.

[80] M Purser. *Introduction to Error Correcting Codes*. Artech House, Boston, 1995.

[81] John Rhodes. *Applications of Automata Theory and Algebra via the Mathematical Theory of Complexity to Biology, Physics, Psychology, Philosophy, and Games*. World Scientific Press, 2009.

[82] Derek J. S. Robinson. *A Course in the Theory of Groups*. Springer, 2nd edition, 1995.

[83] D Schreckling, M Brunato, P Dini, L Dóra, A Faschingbauer, J Golic, G Horváth, F Martinelli, and M Petrocchi. *D4.6: Security in BIONETS*. BIONETS Deliverable, European Commission, 2009. Available from: `http://www.bionets.eu`.

[84] D Schreckling and P Dini. Distributed Online Evolution: An Algebraic Problem? In *IEEE 10th Congress on Evolutionary Computation, Trondheim, Norway, 18-21 May*, 2009.

[85] S Y Shieh, M Ikeda, Y Taya, and C Prives. Dna damage-induced phosphorylation of p53 alleviates inhibition by mdm2. *Cell*, 91:325–334, 1997.

[86] F R Spitz, D Nguyen, J M Skibber, R E Meyn, R J Cristiano, and J A Roth. Adenoviral-mediated wild-type p53 gene expression sensitizes colorectal cancer cells to ionizing radiation. *Clin Cancer Res*, 2:16665–16671, 1996.

[87] O E Staples, J J Hollick, J Campbell, M Higgins, A R McCarthy, V Appleyard, K E Murray, L Baker, A Thompson, S Ronseaux, A M Z Slawin, D P Lane, N J Westwood, and S Lain. Characterization, chemical optimization and anti-tumor activity of a tubulin poison identified by a p53-based phenotypic screen. *Cell Cyle*, 21:3417–3427, 2008.

[88] Y Tang, W Zhao, Y Chen, Y Zhao, and W Gu. Acetylation is indispensable for p53 activation. *Cell*, 133:612–626, 2008.

[89] I Van Leeuwen, A J Munro, I Sanders, O Staples, and S Lain. Numerical and Experimental Analysis of the p53-mdm2 Regulatory Pathway. In Fernando A B Colugnati, Lia C R Lopes, and Saulo F A Barretto, editors, *Digital Ecosystems: Proceedings of the 3rd International Conference, OPAALS 2010*, pages 266–284, Aracaju, Sergipe, Brazil, 22-23 March, 2010. Springer LNICST.

[90] I M M van Leeuwen and S Lain. Chapter 5: Sirtuins and p53. *Adv Cancer Res*, 102:171–195, 2009.

[91] L T Vassilev. p53 activation by small molecules: application in oncology. *J Med Chem*, 48:4491–4498, 2005.

[92] L T Vassilev. Mdm2 inhibitors for cancer therapy. *Trends Mol Med*, 13:23–31, 2007.

[93] L T Vassilev, B T Vu, B Graves, D Carvajal, F Podlaski, Z Filipovic, N Kong, U Kammlott, C Lukacs, C Klein, N Fotouhi, and E A Liu. *In vivo* activation of the p53 pathway by small-molecule antagonists of mdm2. *Science*, 303:844–848, 2004.

[94] H Vaziri, S K Dessain, E Ng Eaton, S I Imai, R A Frye, T K Pandita, L Guarente, and R A Weinberg. hsir2(sirt1) functions as an nad-dependent p53 deacetylase. *Cell*, 107:149–159, 2001.

[95] A Ventura, D G Kirsch, M E McLaughlin, D A Tuveson, J Grimm, L Lintault, J Newman, E E Reczek, R Weissleder, and T Jacks. Restoration of p53 function leads to tumour regression *in vivo*. *Nature*, 445:661–665, 2007.

[96] B Vogelstein, D P Lane, and A J Levine. Surfing the p53 network. *Nature*, 408:307–310, 2007.

[97] S G Williams. Introduction to Symbolic Dynamics. In S G Williams, editor, *Symbolic Dynamics and Its Applications: AMS Short Course, 4-5 January*, 2002.

[98] A Zauberman, D Flusberg, Y Haupt, Y Barak, and M Oren. A functional p53-response intronic promoter is contained within the human mdm2 gene. *Nucleic Acids Res*, 23:2584–2592, 1995.

[99] H Paul Zeiger. Cascade synthesis of finite state machines. *Information and Control*, 10:419–433, 1967. plus erratum.

[100] H Paul Zeiger. Yet another proof of the cascade decomposition theorem for finite automata. *Math. Systems Theory*, 1:225–228, 1967. plus erratum.