Hierarchical Coordinate Systems for Understanding Complexity and its Evolution, with Applications to Genetic Regulatory Networks

Attila Egri-Nagy*
Chrystopher L. Nehaniv
BioComputation and Algorithms Research Groups
School of Computer Science
University of Hertfordshire
College Lane, Hatfield, Herts, AL10 9AB, United Kingdom
Tel: +44-1707-284769, +44-1707-284470
`A.Egri-Nagy,C.L.Nehaniv@herts.ac.uk`

April 23, 2009

Abstract

Beyond complexity measures, sometimes it is worth in addition investigating how complexity changes structurally, especially in artificial systems where we have complete knowledge about the evolutionary process. Hierarchical decomposition is a useful way of assessing structural complexity changes of organisms modeled as automata, and we show how recently developed computational tools can be used for this purpose, by computing holonomy decompositions and holonomy complexity. To gain insight into the evolution of complexity, we investigate the smoothness of the landscape structure of complexity under minimal transitions. As a proof of concept, we illustrate how the hierarchical complexity analysis reveals symmetries and irreversible structure in biological networks by applying the methods to the lac operon mechanism in the genetic regulatory network of *Escherichia coli*.

Keywords: algebraic biology, computational biology, coordinate systems, Krohn-Rhodes theory, finite state automata.

1 Introduction

It has been a common practice in artificial life and biological research to focus on one particular aspect of a living, life-like, or evolutionary system. It is usually the fitness, genome size, etc., or the somehow assessed complexity of individuals. Complexity changes (either increasing or decreasing) may indicate that something significant is happening, but do not reveal what exactly or how it is happening. Did a completely new component appear? Or are the existing components just reused in a novel way? Or has an existing component been duplicated and used in different contexts? These considerations do not entail that analysis should go down to the finest details, although this may be beneficial in some cases. Instead, we propose a method of hierarchical decomposition which gives a coordinate system, from which one has, not only a quantitative complexity measure, but can also read the internal structure of the underlying phenomenon with arbitrary resolution at different levels of abstractness appropriate for the system in question. This approach allows one to gain mathematical insight into the structure of dynamical hierarchies (cf. [18]). Note that whether or not hierarchies are actually present in real world processes, they are very often invaluable tools for understanding the structure of a complex system.

The mathematical theory behind this is the algebraic hierarchical decomposition theory of finite state automata, Krohn-Rhodes Theory [16, 5, 25], and is introduced in Section 2. The idea of using hierarchical decompositions as cognitive tools for fostering our (or any other intelligent agent's) understanding was proposed several times [25, 21], but now is closer to fulfillment. For any evolved or designed system amenable to modeling as a finite automaton, Krohn-Rhodes Theory provides natural complexity measures [15, 28, 21] which are related to but distinct from Kolmogorov complexity, while giving additional insight for understanding complex structure (see Section 3).

For forty years there had been no computational implementation for the mathematically demonstrated hierarchical decomposition of automata. Although in the electronic circuit industry there are many different decomposition methods and implementations, they are not hierarchical since there are several physical constraints on circuit design and the hierarchical (cascade) composition appears not to be the most efficient in terms of power consumption, area and delay minimization [4]. However, recently the authors have computationally implemented two methods for the holonomy decomposition [9, 8, 7], a particular version of the Krohn-Rhodes decomposition, which is applied here to understanding complexity. This method is applicable to yield a hierarchical decomposition of any finite automaton, including models of organisms in evolving populations (whether natural or artificial). Section 4 introduces how these computational tools can be used to create hierarchical coordinate systems for understanding any system modeled in this way. Section 5 illustrates the application of the holonomy decomposition technique to investigating variability in complexity on the landscape of automata under minimal variations (i.e. within a small neighborhood of a given automaton). Section 6 applies the method to understanding complexity in a genetic regulatory network model of the bacterium *E. coli* revealing symmetry and irreversible structure, while the final section discusses other possible future applications and scalability issues for complexity computation.

2 Hierarchical Decomposition: The Krohn-Rhodes Theory

Here we present the very basic underlying ideas of algebraic hierarchical decomposition of finite state automata. We use the minimum amount of mathematical notation here. For precise definitions and details, see [17, 21, 5].

2.1 Reversible and Irreversible Processes

There are two different kinds of computational operations: reversible and irreversible ones. For instance, if we move some content of the memory to another empty location, that is reversible, since we can move it back. But if we overwrite a nonempty part of the memory, then this is irreversible, since there is no way to restore the previously stored data. Closer to a formal definition we can say that irreversible processes reduce the size of the set of possible future states, while reversible ones do not.

Algebraically the distinction is more immediate. We view a system we wish to understand as having a set of possible *states* A, upon which various possible transformations (called "inputs", "events" or "input symbols") can act which alter its states.[1] A function $f : A \to A$ of a set A is called a permutation (reversible) if it is one-to-one and onto; otherwise, it must collapse elements (some $a \in A$ is an image of more than one element, i.e. the image of A under f is a proper subset of A), therefore it is irreversible. A *permutation group* is a set G of invertible mappings together with the state set A on which the mappings act. A *transformation semigroup* (A, S) with state set A and transformations S has a similar structure, but S consists of general transformations, not necessarily just permutations. Krohn-Rhodes Theory allows one to work with transformation semigroups rather than finite automata, and apply abstract algebra to yield results on the natural hierarchical structuring "hidden" in these automata. The elements of the semigroup are the transformations of the state set generated by the input symbols. This way the problems in automata theory are transferred into the algebraic domain, where there are a rich mathematical theory and

[1] In elementary Krohn-Rhodes Theory, the state set A is assumed finite and the transformations are deterministic.

rigorous complexity measures.

2.2 The Prime Decomposition Metaphor

For explaining Krohn-Rhodes Theory, the best way is to present it by a metaphor. Basically we do the same for complex systems as the decomposition into prime factors does for natural numbers, but instead of integer numbers we do it for more complicated structures, namely finite state automata (considered as transformation semigroups). The similarities can be summarized the following way:

	Natural Numbers	**Finite Automata**
Building Blocks	Primes	Flip-flop Automaton Permutation Automata
Composition	Multiplication	Wreath Product
Precision	Equality	Division, Emulation
Uniqueness	Unique	Different Decompositions

The basic building blocks are (1) the simple[2] 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.[3]

The way of putting together the components, the so-called *cascaded* or *wreath* product, is hierarchical and no feedback is allowed from deeper levels to upper levels (see Fig. 1). The usefulness of this special type of composition is due to the following special properties of hierarchy that render the composed structure manipulable and comprehensible:

[2]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. See, e.g. [26].

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

- Generalization and specialization are natural operations realized by taking subsets of levels in either direction up or down the hierarchy.
- Information flow between levels is restricted, avoiding problems of feedback.

Note that any number of parallel, non-interacting components are allowed on any hierarchical level.

The hierarchical composition is a proper balance between the two conflicting requirements: having a nice, comprehensible structure and possessing the expressive power to construct any arbitrary automaton (see Fig. 2). 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. [6]), but such a construction generally provides no insight into its structure. 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.

2.3 Coordinates and Hierarchical Dependence

Hierarchical decompositions provide a coordinate system for the original system that has been described as an automaton. By a coordinate system we mean a notational system (in the broadest possible sense), with which we can address the components and their relations in a decomposition, thus gaining a convenient way for grasping the structure of the studied phenomenon. For each coordinate position we have a transformation semigroup (the corresponding component of the wreath product), and elements of its state set are the possible values for that position. Due to its hierarchical nature the order of the coordinates does matter. What happens on deeper levels depends not only on input to the system but also on the states of the levels above.

A simple non-trivial example to describe *hierarchical dependence* is a bidirectional counter. Imagine a device which keeps track of how many times you press a button, where you also have two other buttons to set the operating mode. For instance to count and double-check the number of passengers on an airplane while walking along the aisle, you start from zero in adding mode, count, and then as a check whether the resulting number is the correct value, you switch to subtracting mode and count again, but this time downwards, until you reach zero again. The operation of this device can be represented with the following simple coordinate system on its states:

$$(n, \mathrm{mode}),$$

where n is the current tally and the possible modes $+$ and $-$ correspond to adding and subtracting. The mode coordinate is the top level of the hierarchy. The buttons provide three operations: counting c, switching to adding mode m_+, and switching to subtracting mode m_-. For instance, the result of each elementary operation is exemplified as follows:

$$(9,+)\cdot c = (10,+)$$

$$(9,+)\cdot m_- = (9,-)$$

$$(9,+)\cdot m_+ = (9,+)$$

$$(9,-)\cdot c = (8,-)$$

Hierarchical dependence here is clear: the counting operation does different things (adding or subtracting 1) depending on the top level coordinate (the right coordinate giving the current mode); but this dependence is only one way: the state of the tally count (left coordinate) never influences the effect of the basic transformations on the mode coordinate.

3 The Number of Hierarchical Levels as a Complexity Measure

We consider complexity also as a structure not just as a number: we study the hierarchical decomposition as a coordinate system. Beyond the hierarchical structure we also get a single-valued measure for complexity by hierarchical decomposition, namely the number of hierarchical levels. The number of levels in such a decomposition is thus a raw complexity measure and structure of the hierarchical coordinate system describes the components and their interaction giving rise to this complexity. But since decompositions are not unique, this measure can be defined differently depending on the decomposition algorithms employed.

3.1 Group Complexity

One of the most natural questions about hierarchical decompositions concerns the length of the cascaded product. What is the shortest possible decomposition for a given automaton? The original mathematical formulation of group complexity (also known as *Krohn-Rhodes complexity*) counts the number of alternations between group and aperiodic components (those composed of flip-flops) in hierarchical decompositions of a given automaton. The smallest possible number of alternations is the *group complexity* of the original automata [15, 28]. This value as a complexity measure is amenable to a system of axioms [20, 21] and also can be used for assessing the evolvability of evolutionary systems [19]. However, determining the size of the shortest such decomposition turns out to be a very difficult problem. Unlike Kolmogorov complexity, group complexity is believed to be algorithmically decidable, although no correct proof of this has yet appeared in print; and even if it is decidable, it still will not likely be practically computable.

3.2 Holonomy Complexity

Since we are interested in practical applications to the evolution of complexity in natural and artificial living systems, and not necessarily looking for the shortest decompositions but rather feasible decompositions that are computationally accessible, we promote a particular decomposition method. The *holonomy decomposition* method for obtaining a Krohn-Rhodes decomposition [29, 30, 12, 11, 5] originates from ideas in computer science for coding nests of sets [29, 30]. It identifies algebraically salient subprocesses within an automaton, especially permutation group structures, by investigating transformations of certain subsets of the state set of the automaton. The hierarchical structure also reveals the flow of the irreversible computations possible in the automaton, as the subsets of the original state set (induced by the transformations) represent reductions of the set of future possible states.

Our open-source computational tool `JGRASP` [8] allows one not only to calculate the number of levels in this decomposition, i.e. calculate *holonomy complexity*, but produces the holonomy decomposition coordinate system with appropriate reversible and irreversible components for understanding the complexity in the original system, as illustrated in the next section.

4 Hierarchical Coordinate Systems as Tools for Understanding

The general strategy of using coordinate systems to understand complex systems can be summarized in the following table.

Complex phenomenon

↓

Finite description (automaton model)

↓

Hierarchical coordinate system for understanding the model

↓

Coordinate system for understanding and
manipulating the original phenomenon.

Modelling complex phenomena using automata is a powerful but delicate method for beginning to understand them [25, 13, 22]. Here we skip this first step and start with finite state automata. Let's suppose we have an automaton and we do not really know what it is doing (although by knowing its generators we fully describe it implicitly), as in the example shown in Fig. 3, which is the state transition graph of a randomly generated automaton. Is it doing some complex computation? In order to find this out we calculate its holonomy decomposition. The holonomy method finds constituent components by analysing how the transformations act on certain sets of subsets of the state set. The decomposition can be automatically generated by our open-source tool `JGRASP` [8]. By studying the hierarchical structure we find that the automaton can be emulated by a cascaded automaton with two levels (for details and visualization see Figures 4-6). Now if we ask the question, 'What does the automaton do roughly?', then we can answer very easily just by looking at the top level (Fig. 5). We have three abstracted states there and the component is not a reversible one, so it behaves like a memory that can be set into one of 3 abstract states: $\{3,4,5\}$, $\{2,3\}$, and $\{1\}$. Going further down to level 1 we find that depending on the abstract state above we either have a reversible component, a one bit memory, or a degenerate garden-of-eden state.[4] The actual reversible component is a permutation of three states of the original automaton, where two of them are transposable and the other is fixed (Fig. 6).[5] If the one-bit memory is active, a state from $\{2,3\}$ encodes the one bit of information.

This illustrates the idea of having a coordinate system for understanding the computation of an automaton.

5 A Glimpse into Complexity on the Vast Landscape of Automata Decompositions

Our method guarantees that the decomposition retains and highlights the important features of the original system. To get insight into the structure of the space of automata and their decompositions, we check how the complexity changes – in terms number of levels in the decomposition (see Section 3) – as we smoothly perturb the automaton, reflecting the ruggedness or smoothness of the complexity landscape. This gives insight into the evolutionary landscape of automata where minimal variations (or 'mutations') consist of (1) redirecting an arrow in the state transition graph, or (2) adding or deleting a disconnected state – i.e. a state with no transitions from or to any state other than itself. Note that all finite automata are mutually reachable by sequences of transitions of these types. As variations of the latter type by themselves cannot essentially

[4]A garden-of-eden state is a state that can never be returned to.

[5]In more general cases, permutation groups in the holonomy decomposition act on a set of abstracted, higher-level states, i.e. they permute a set of subsets of the full state set of the automaton (not necessarily a set of singletons). Such permutation groups are called *holonomy groups* and their constituent permutations are called *holonomy permutations.*

change complexity as measured by the holonomy decomposition, we investigate only the first type. Nevertheless, the minimal variations of type 2, clearly open up the space of future possibilities and increased complexity to evolution.

5.1 Rugged Variability

We start with an automaton with three input symbols (Fig. 7). This automaton has a holonomy decomposition of length 8 (i.e. with 8 hierarchical levels). We generate all the automata that are just one mutation far from this automaton. Mutation here is changing one single transition for an input symbol to another value, i.e. changing the target of one single arrow in the state transition diagram. Therefore we have 168 automata in the automaton's closest neighborhood. Fig. 8 shows the distribution of the number of hierarchical levels of the decompositions in this neighborhood. From this we can see that small changes may yield completely different decompositions, as the number of hierarchical levels varies from 5 to 18.[6]

5.2 Smooth Variability

Now we demonstrate that gradual changes in holonomy complexity and structure are also possible for mutations of type 1. Intuitively, these changes should have only local effects, and since the reversible computations correspond to cycles in the state transition graph, they should not create or destroy cycles (compare Figures 9 & 10). Fig. 10 shows by example that such gradual change is possible. Moreover, it is apparent from Fig. 8 that a significant fraction –

[6]Note that, although no theorem has been proved on holonomy complexity change under single-step mutations, these holonomy complexity values are within the range one would expect from the theoretically demonstrated limits for smooth evolutionary change on complexity for the Krohn-Rhodes complexity measure [20]. However it is easy to construct examples – such as long cycles – in which a single change on arrows can result in a change from 1 level to n hierarchical levels in the holonomy decomposition (see Fig. 9). (This changes group complexity, in contrast, from 1 to 0.)

though not a majority – of mutations to the automaton in Fig. 7 leave the holonomy complexity (which has value 8) unchanged.

Evolution operating in this high dimensional landscape of automata whose local neighborhoods are defined by mutating simple state-transitions thus has access to both ruggedly sharp and gradual types of changes.

6 Case Study: The Lac Operon in *E. coli*

Now we apply hierarchical decomposition to a well-known example. *Escherichia coli*, the "workhorse" bacterium of microbiology, can metabolize glucose and lactose as well, but it prefers glucose. Therefore lactose is metabolized only when glucose is absent and lactose is available. In all other cases the expression of the structural genes for the enzymes of lactose metabolism are suppressed. This gene regulatory mechanism, the lac operon, is well understood now, and it is the canonical example of prokaryotic gene regulation (see e.g. [23, 24]).

6.1 Complexity of the Lac Operon

We would like to apply the decomposition to the lac operon mechanism. First we need a finite state automaton description of this gene regulation mechanism. Here we can use a very simple Boolean network model of the lac operon mechanism in *E. coli*, as originally suggested by Stuart Kauffman [13].[7] The state transition graph of the automaton can be seen in Fig. 11. Again, we have an automaton and we wish to understand what it is does. Although by knowing its generators or transition diagram one can fully describe it implicitly, this gives us no insight into its structure and no capacity to abstractly describe various levels of its computation. What kind of computation does it perform? And how

[7]We are grateful to George F. Estabrook for providing us with the details of Kauffman's model.

does the performed computation relate to the original system? It is difficult to see from the state transition graph of the automaton (although in this very simple case it is not impossible). Calculating its holonomy decomposition yields a hierarchy with 5 levels (Fig. 12). A closer look shows that the top 3 levels are transient; they include states that occur only for short time (i.e. exactly during the depletion of lactose), and they may lead to a shut-off attractor state n (in the continued absence of lactose). The remaining two levels show the presence of a non-trivial symmetry group operating within the sets of two attractor metabolic cycles (in the presence of lactose) (Fig. 13): On level 2 we have two abstract states representing the absence and the presence of lactose, if lactose is present then the system is in the nontrivial group component of level 1, i.e. reflecting a cyclic process of metabolizing lactose, otherwise the system is at the fixed point attractor where lactose metabolism is shut off.

The result obtained here is not as interesting for novelty as it is for showing that automatic hierarchical coordinatization can find essential structure in complex biological systems. We knew beforehand about the metabolic cycle, but this gives proof of the concept that hierarchical coordinatization can yield a concise and easy to understand description of an unanalyzed complex system, revealing natural internal symmetries and irreversibility structure.

6.2 Ensemble Approach

This biological example suggests another interpretation for the hierarchical holonomy decomposition, based on ideas borrowed from statistical mechanics. In this *ensemble approach*[8], we consider the state transitions of many copies of the same automaton, not just one individual automaton. For example, the state of a lac operon in each cell in a large population of *E. coli* cells growing in Petri

[8]This idea is due to John L. Rhodes [25] following physicist Erwin Schrödinger [27].

dish may be modeled as a copy of the automaton in Figure 11. The subset of states actually occurring in the population corresponds to sets appearing in the holonomy decomposition. One uses the decomposition as a map with arbitrary scale (considering subsets of the hierarchical levels) for the global states of the ensemble of cells. This approach is appealing biologically and also removes some of the difficulties originating from a discrete modeling approach.

Clearly, the global state of the ensemble can be defined in many different ways (e.g. the set of observed states, or their frequency distribution), and the choice of an "update rule" (how the transformations are applied to the individual automata, the members of the ensemble), if not synchronous, may potentially change the behaviour of the ensemble. However, even before these issues are studied further, the ensemble approach can be used as a guiding metaphor for applying the holonomy decomposition to understanding complex systems.

In the lac operon example, being in the state represented by level 1 in the cascaded product (the metabolic cycle) can be interpreted as that in the many cells of the ensemble after manipulating and observing them, the set of observed states is exactly $\{a, b, c, d, e, f, g, h\}$. Any input from the holonomy group component permutes this set of cell states in the ensemble and thus maintains it as an invariant set of possible future states taken by the members of the ensemble (assuming that all members of the ensemble always receive the same inputs at the same time). Figures 5 and 6 can be interpreted similarly.

7 Future Work and Discussion

Since we would like to apply these methods to real-world problems the algorithms should be scalable. Currently we are working on an incremental version of the algorithm, which starts at the top level and goes down to decompose further levels when they are feasible. This way we get some information about the hierarchical structure immediately, instead of trying to calculate the whole decomposition all at once, which may fail due to combinatorial complexity. However, the current software tools are already far beyond the capabilities of the human's pen and paper method.

Beyond increasing scalability it would be desirable to apply this tool for understanding natural and artificial genetic regulatory networks (GRNs) [3, 14]. One strategy is to represent GRNs with Petri-nets, which can be easily converted to finite state automata. This requires some theoretical preparatory work, since Petri-nets can be converted to finite state automata in many different ways, which might change the resulting model. Another strategy would also be to apply Crutchfield's ϵ-machine reconstruction [2] (with a finite time window size) for a series of observations of real or artificial GRNs, which again yields a finite automaton. It might also be intriguing to relate the holonomy decomposition to graph properties (spectrum, diameter, connectivity, etc.; see e.g. [1]) of the state transition diagram, however the fact that the decomposition depends heavily on cycles labelled by powers of input words [10] and may change under relabelling suggests that such an approach could be challenging to develop.

It will be important to relate evolution in the landscape of automata as described above to evolution of biological organisms with their characteristic types of genetic variability. Our basic assumption is that the individuals or even the whole evolutionary system can be adequately described by finite state automata. This is ensured for any *in silico* experiment, since the computer on which it is carried out is a (huge) finite state automaton. One can argue whether our assumption is suitable for real biological systems, or whether by using discrete non-stochastic models we abstract away important layers of the working machinery. However, since many researchers are generating descriptive but analysed finite state models of biological systems, our approach has a wide range of possible applications. Current work on the computational implementations is aimed at allowing one to apply these methods to understanding the complexity of particular example biochemical and genetic regulatory networks.

References

[1] Chung, F. R. K. (1997) *Spectral Graph Theory*. American Mathematical Society.

[2] Crutchfield, J. P. (1994) The calculi of emergence: Computation, dynamics, and induction. *Physica D*, **75**, 11–54.

[3] Davidson, E. H. (2006) *The Regulatory Genome: Gene Regulatory Networks in Development and Evolution*. Academic Press.

[4] Devadas, S. & Newton, A. R. (1989) Decomposition and factorization of sequential finite state machinces. *IEEE Transactions on Computer-Aided Design*, **8**, 1206–1217.

[5] Dömösi, P. & Nehaniv, C. L. (2005) *Algebraic Theory of Finite Automata Networks: An Introduction*, chap. 3, The Krohn-Rhodes and Holonomy Decomposition Theorems. SIAM Series on Discrete Mathematics and Applications.

[6] Dömösi, P. & Nehaniv, C. L. (2005) *Algebraic Theory of Finite Automata Networks: An Introduction*. SIAM Series on Discrete Mathematics and Applications.

[7] Egri-Nagy, A. (2005) *Algebraic Hierarchical Decomposition of Finite State Automata – A Computational Approach*. Ph.D. thesis, University of Hertfordshire, School of Computer Science, United Kingdom.

[8] Egri-Nagy, A. & Nehaniv, C. L. (2003), GrasperMachine, Computational Semigroup Theory for Formal Models of Understanding. `(http://graspermachine.sf.net)`.

[9] Egri-Nagy, A. & Nehaniv, C. L. (2004) Algebraic hierarchical decomposition of finite state automata: Comparison of implementations for Krohn-Rhodes Theory. *Conference on Implementations and Applications of Automata CIAA 2004, Lecture Notes in Computer Science*, **3317**, 315–316.

[10] Egri-Nagy, A. & Nehaniv, C. L. (2005) Cycle structure in automata and the holonomy decomposition. *Acta Cybernetica*, **17**, 199–211, [ISSN: 0324-721X].

[11] Eilenberg, S. (1976) *Automata, Languages and Machines*, vol. B. Academic Press.

[12] Ginzburg, A. (1968) *Algebraic Theory of Automata*. Academic Press.

[13] Kauffman, S. A. (1969) Metabolic stability and epigenesis in randomly connected genetic nets. *Journal of Theoretical Biology*, **22**, 437–467.

[14] Knabe, J. F., Nehaniv, C. L., Schilstra, M. J., & Quick, T. (2006) Evolving biological clocks using genetic regulatory networks. Rocha, L. M., Yaeger, L. S., Bedau, M. A., Floreano, D., Goldstone, R. L., & Vespignani, A. (eds.), *Proceedings of the Artificial Life X Conference*, pp. 15–21, MIT Press.

[15] Krohn, K. & Rhodes, J. (1965) Results on finite semigroups derived from the algebraic theory of machines. *Proc. Natl. Acad. Sci. U.S.A.*, **53**, 499–501.

[16] Krohn, K. & Rhodes, J. (1965) Algebraic theory of machines. I. Prime decomposition theorem for finite semigroups and machines. *Transactions of the American Mathematical Society*, **116**, 450–464.

[17] Krohn, K., Rhodes, J. L., & Tilson, B. R. (1968) The prime decomposition theorem of the algebraic theory of machines. Arbib, M. A. (ed.), *Algebraic Theory of Machines, Languages, and Semigroups*, chap. 5, pp. 81–125, Academic Press.

[18] Lenaerts, T., Chu, D., & Watson, R. (2005) Dynamical hierarchies. *Artificial Life*, **11**, 403–405.

[19] Nehaniv, C. L. (2000) Measuring evolvability as the rate of complexity increase. Maley, C. C. & Boudreau, E. (eds.), *Artificial Life 7 Workshop Proceedings*, pp. 55–57.

[20] Nehaniv, C. L. & Rhodes, J. L. (1999) On the manner in which biological complexity may grow. *Mathematical and Computational Biology, Lectures in the Life Sciences*, vol. 26, pp. 93–102, American Mathematical Society.

[21] Nehaniv, C. L. & Rhodes, J. L. (2000) The evolution and understanding of hierarchical complexity in biology from an algebraic perspective. *Artificial Life*, **6**, 45–67.

[22] Peleg, M., Rubin, D., & Altman, R. B. (2005) Using Petri Net Tools to Study Properties and Dynamics of Biological Systems. *Journal of the American Medical Informatics Association*, **12**, 181–199.

[23] Ptashne, M. (1992) *A Genetic Switch: λ and Higher Organisms*. Blackwell Publishers, 2nd edn.

[24] Ptashne, M. & Gann, A. (2001) *Genes and Signals*. Cold Spring Harbor Laboratory Press.

[25] Rhodes, J. L. (to appear 2007) *Applications of Automata Theory and Algebra via the Mathematical Theory of Complexity to Finite-State Physics, Biology, Philosophy, Games, and Codes*. World Scientific Press, foreword by Morris W. Hirsch, edited by Chrystopher L. Nehaniv (Original version: University of California at Berkeley, Mathematics Library, 1971).

[26] Robinson, D. J. S. (1995) *A Course in the Theory of Groups*. Springer, 2nd edn.

[27] Schrödinger, E. (1967) *Statistical Thermodynamics*. Cambridge University Press.

[28] Tilson, B. (1976) Complexity of semigroups and morphisms. Eilenberg, S. (ed.), *Automata, Languages and Machines*, vol. B, chap. XII, Academic Press.

[29] Zeiger, H. P. (1967) Cascade synthesis of finite state machines. *Information and Control*, **10**, 419–433, plus erratum.

[30] Zeiger, H. P. (1968) Cascade decomposition using covers. Arbib, M. A. (ed.), *Algebraic Theory of Machines, Languages, and Semigroups*, chap. 4, pp. 55–80, Academic Press.

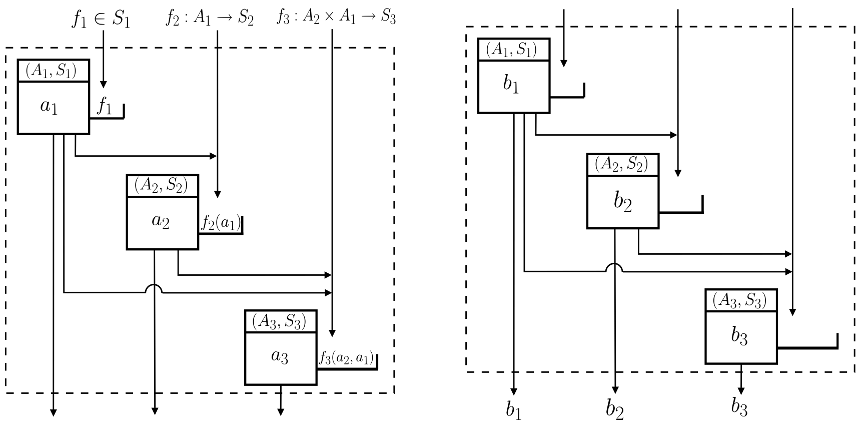


Figure 1: Example of a 3-level coordinate system composed using the cascaded/wreath product of component transformation semigroups (A_n, S_n), $n \in \{1, 2, 3\}$. The resulting composed automaton is enclosed in dashed lines; both its input and output are 3-tuples. Left: For a state transition in the wreath product $(A_3, S_3) \wr (A_2, S_2) \wr (A_1, S_1)$, the input transformation (f_3, f_2, f_1) is applied to state (a_3, a_2, a_1) yielding $(b_3, b_2, b_1) = (a_3 \cdot f_3(a_2, a_1), a_2 \cdot f_2(a_1), a_1 \cdot f_1)$. The "trays" visualize how at each level i the components of the input (dependency functions f_i) are evaluated according to hierarchical dependence on the states at higher levels. The resulting transformations $f_i(a_{i-1}, \ldots, a_1) \in S_i$ are then applied to transform the state component within level i. Note that the applications of these functions happen simultaneously; their arguments are the previous states of other components, therefore there is no need to wait for the other components to calculate their new states. Right: The new state (b_1, b_2, b_3) is (without loss of generality) the output of the automaton. Projection onto initial coordinates is a structure-preserving mapping (homomorphism).

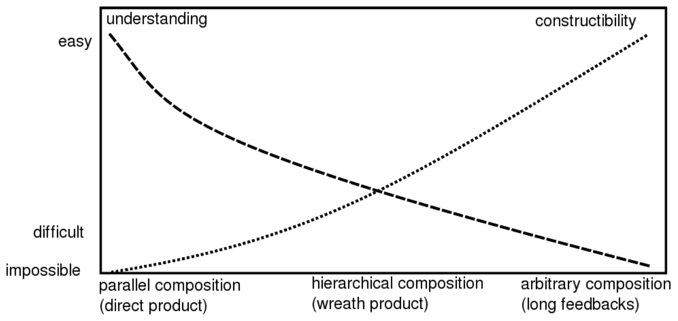


Figure 2: Schematic diagram of the position of the wreath product relative to alternative decomposition methods.

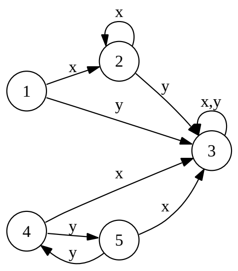


Figure 3: State transition diagram of a randomly generated automaton $\mathcal{R}$ with 5 states and transition arrows for two input symbols x and y. The arrows encode state changes following the transformations $x = \left(\begin{smallmatrix}1&2&3&4&5\\2&2&3&3&3\end{smallmatrix}\right)$, $y = \left(\begin{smallmatrix}1&2&3&4&5\\3&3&3&5&4\end{smallmatrix}\right)$, where the upper row lists the states of the automaton and the lower row gives the corresponding resulting states after the transformation.

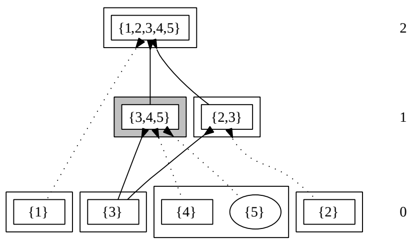


Figure 4: The structure of the holonomy decomposition of the automaton $\mathcal{R}$. This is the *tiling picture* of the decomposition, i.e. it is the structure of how the state set is recursively covered with its subsets. The numbers on the right denote the hierarchical levels (the level 0 is included to show the states of the components on level 1, it does not appear as a separate hierarchical level in the decomposition). The nodes are subsets of the state set, outer rectangular nodes represent the components of the decomposition. Shaded components denote the existence of some reversible computation possible in the system. The arrows going into the component come from the component's states (solid and dotted arrows indicating state sets of two types: solid line means that it is a real image of the parent state set and dotted means that the child node comes from somewhere else). On level 1 we have parallel, non-interacting components.

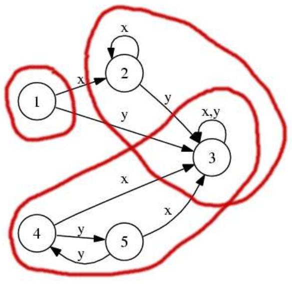


Figure 5: The states of the top level component of the decomposition of automaton $\mathcal{R}$ are overlapping subsets of the state set.

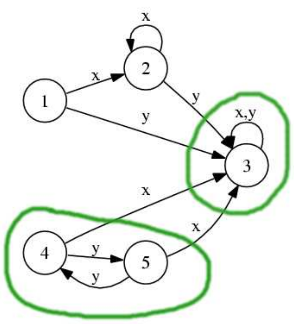


Figure 6: The identified reversible computation at the level 1, the level below the top level, of the holonomy decomposition of automaton $\mathcal{R}$. The input y permutes the 3 circled states, transposing states $\{4\}$ and $\{5\}$ while fixing the state $\{3\}$.

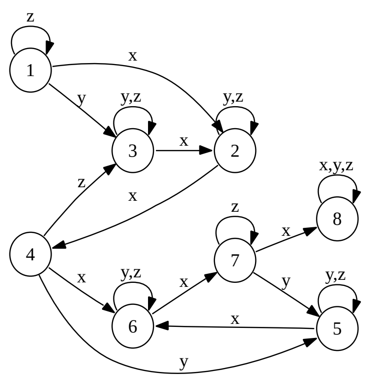


Figure 7: A finite state automaton with three input symbols representing the transformations $x = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 2 & 4 & 2 & 6 & 6 & 7 & 8 & 8 \end{pmatrix}$, $y = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 3 & 2 & 3 & 5 & 5 & 6 & 5 & 8 \end{pmatrix}$, $z = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 3 & 5 & 6 & 7 & 8 \end{pmatrix}$.

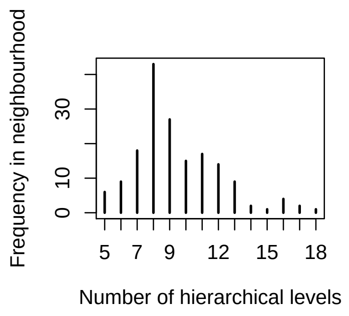


Figure 8: Frequencies of holonomy complexity values for the one mutation neighborhood. The height of the decomposition of the original automaton is 8.

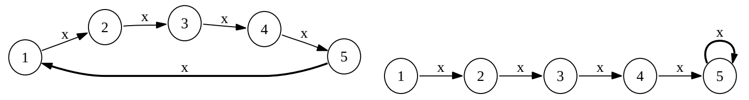


Figure 9: An example of one mutation, where redirecting one arrow completely changes the decomposition. The left automaton has a decomposition with only hierarchical one level, consisting of a single reversible component (the whole cycle). The automaton on the right, where only the target of the arrow leaving state 5 has been redirected, has 4 hierarchical levels in its decomposition as it has become completely irreversible.

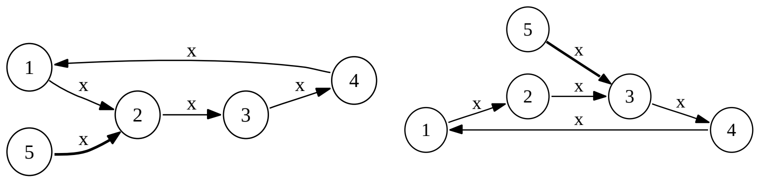
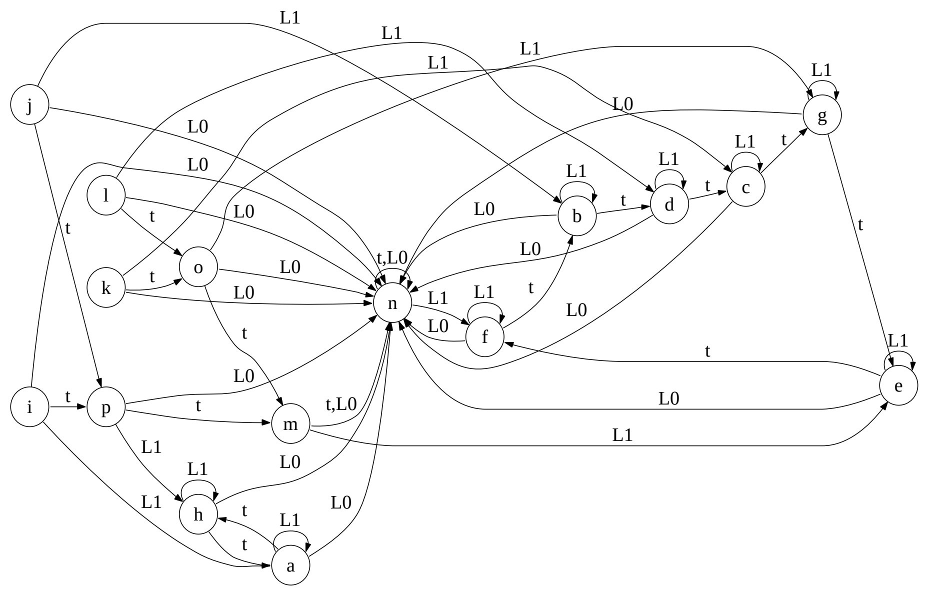


Figure 10: An example of a “harmless” mutation: changing the target of the arrow coming out from state 5. The change does not affect the cycle, and does not introduce any new state set reduction, thus the structure of the decomposition is preserved. Clearly, if further mutations are applied these neutral differences can lead to different decompositions.



STATE	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
A	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
Op	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
ZYA	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
lactose	present								absent							

Figure 11: The automaton derived from Boolean network model of the lac operon mechanism in *E. coli.* Transition L1 denotes the introduction of lactose, L0 the depletion of lactose, and *t* labels transitions due to passage of time. The states are defined by Boolean combinations of the presence or absence of biochemical components: **A**: allolactose is an isomer of lactose, **Op**: the repressor molecule, **ZYA**: the structural genes for the enzymes needed for lactose metabolism. Here, 1 means that the molecule is present/active or the gene is expressed, 0 is for the absence/inactivity. The transformations of the system are given by all finite sequences of transitions from L1, L0, and *t*.

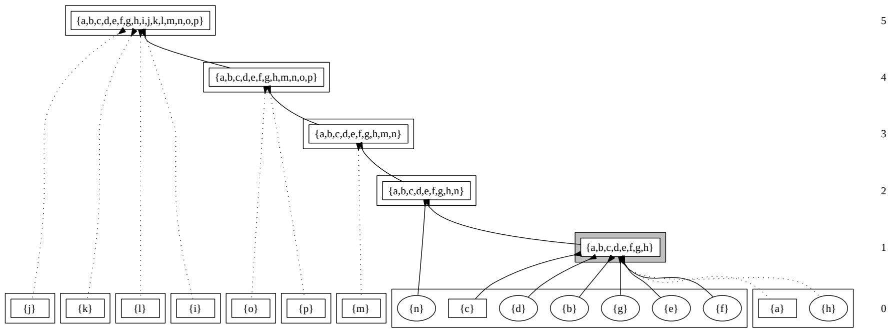


Figure 12: The holonomy decomposition of the Boolean network model of the lac operon mechanism in *E. coli*. The top 3 levels are transient.

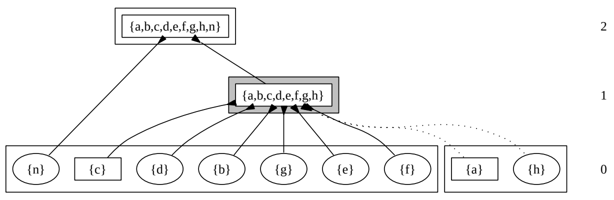


Figure 13: A closeup of the most important (non-transient) levels, levels 2 and 1, in which a non-trivial cyclic group of permutations (shaded component) operates, reflecting the lactose processing metabolic cycle.