Algebraic Hierarchical Decompositions of Finite State Automata – A Computational Approach

Attila Egri-Nagy

A thesis submitted in partial fulfillment of the
requirements of the University of Hertfordshire
for the degree of

Doctor of Philosophy

The programme of research was carried out in the School of
Computer Science, Faculty of Engineering and Information Sciences
University of Hertfordshire.

November 11, 2005

To the first human, who took a piece of stone and used it as a tool...

Acknowledgments

I would like to thank my supervisors, Chrystopher Nehaniv for showing the amusements of doing mathematics together while moving from a restaurant to a cafe and back to a restaurant again endlessly, Bruce Christianson for his laconic but powerful comments, Pál Dömösi for helping my way into an academic career.

I also would like to thank my 'transitive' supervisor, John Rhodes for giving us enough interesting questions to think about.

I would like thank all the kind people closely around me, who might have found it strange and probably annoying when I was immersed in the world of mathematical symbols.

Finally I would like to thank my violin, a Strad model from Transylvania, for using different neurons in my brain, giving a rest for the parts occupied with mathematics.

ATTILA EGRI-NAGY

Abstract

The theory of algebraic hierarchical decomposition of finite state automata is an important and well developed branch of theoretical computer science (Krohn-Rhodes Theory). Beyond this it gives a general model for some important aspects of our cognitive capabilities and also provides possible means for constructing artificial cognitive systems: a Krohn-Rhodes decomposition may serve as a formal model of understanding since we comprehend the world around us in terms of hierarchical representations. In order to investigate formal models of understanding using this approach, we need efficient tools but despite the significance of the theory there has been no computational implementation until this work.

Here the main aim was to open up the vast space of these decompositions by developing a computational toolkit and to make the initial steps of the exploration. Two different decomposition methods were implemented: the $V \cup T$ and the holonomy decomposition. Since the holonomy method, unlike the $V \cup T$ method, gives decompositions of reasonable lengths, it was chosen for a more detailed study.

In studying the holonomy decomposition our main focus is to develop techniques which enable us to calculate the decompositions efficiently, since eventually we would like to apply the decompositions for real-world problems. As the most crucial part is finding the the group components we present several different ways for solving this problem. Then we investigate actual decompositions generated by the holonomy method: automata with some spatial structure illustrating the core structure of the holonomy decomposition, cases for showing interesting properties of the decomposition (length of the decomposition, number of states of a component), and the decomposition of finite residue class rings of integers modulo n.

Finally we analyse the applicability of the holonomy decompositions as formal theories of understanding, and delineate the directions for further research.

Contents

1 Introduction **1**

1.1 The Concept 1

1.1.1 Models 1

1.1.2 Decompositions 2

1.1.3 Hierarchies 2

1.1.4 Coordinate Systems 3

1.1.5 Aspects Do Matter 3

1.2 The Substrate 4

1.2.1 Automata 4

1.2.2 Semigroups 4

1.2.3 Emulation 4

1.2.4 Finiteness, Computational Complexity 5

1.3 Research Questions and Motivations 5

1.3.1 Feasibility 5

1.3.2 Exploration and Exploitation 6

1.3.3 Formal Models of Understanding 6

1.4 Roadmap 7

2 Mathematical Preliminaries and Notations **8**

2.1 Semigroups and Groups 8

2.1.1 Semigroups 8

2.1.2 Groups 9

2.1.3 Transformations and Permutations 9

2.1.4 Green's Relations 10

2.1.5 Homomorphisms 10

2.1.6 Words and the Free Semigroup. 11

2.2 Finite State Automata 12

2.3 Wreath Product 12

2.4 Examples 15

2.4.1 Faces of an Automaton 15

2.4.2 Building a Modulo 4 Counter Hierarchically 17

2.5 Summary 18

3 The Krohn-Rhodes Theory **19**

3.1 The Prime Decomposition Metaphor . . . 19
3.2 The Building Blocks . . . 20
3.3 Wiring the Components . . . 20
3.4 The Krohn-Rhodes Prime Decomposition Theorem . . . 21
3.5 Historical Remarks . . . 21
3.6 Summary . . . 22

4 The $V \cup T$-technique **23**

4.1 The Iterative Construction . . . 23
4.2 Results . . . 25
4.2.1 The Full Transformation Semigroup . . . 25
4.2.2 More Extreme Examples . . . 25
4.3 Summary . . . 26

5 The Holonomy Decomposition **27**

5.1 Holonomy Decomposition Theorem . . . 27
5.2 Relations of the Extended Set of Images . . . 28
5.2.1 The Extended Set of Images . . . 29
5.2.2 Inclusion . . . 29
5.2.3 Image Relation . . . 29
5.2.4 The Subduction Relation and the Skeleton . . . 29
5.2.5 Equivalence Classes . . . 30
5.2.6 Tiles and Tiling . . . 30
5.2.7 Tile Chains . . . 31
5.2.8 Height of a Subset . . . 31
5.3 Components . . . 31
5.3.1 Holonomy Groups . . . 31
5.3.2 Holonomy Permutation-Reset Transformation Semigroups . . . 32
5.4 Mappings on $\mathcal{I}$ . . . 32
5.4.1 Isomorphisms of Holonomy Groups within a Subduction Equivalence Class . . . 32
5.4.2 Moving within an equivalence class . . . 34
5.5 Lifting the State Set . . . 35
5.5.1 Successive Approximation of States . . . 35
5.5.2 Lifting the States . . . 36
5.6 Lifting the Semigroup . . . 36
5.6.1 Lifting Transformations . . . 36
5.6.2 Verifying the division . . . 37
5.6.3 Dependency Functions . . . 39
5.6.4 The Circuitry of the Wreath Product . . . 39
5.7 Examples . . . 40
5.7.1 The Tricks of Tiling . . . 40

5.7.2 Cross Level Tiles 41
5.7.3 Nonimage Tiles 42
5.7.4 Strict Subduction for Sets with the Same Cardinality 43
5.7.5 Tile Chains 43
5.8 Summary 45

6 Implementational Details of the Holonomy Decomposition 46
6.1 Related Software Packages 46
6.2 Representational Issues 47
6.3 Trivial Implementation Using Brute Force Enumeration . . . 48
6.4 Examples 49
6.4.1 Generating Images 49
6.4.2 Deciding Subduction Relation 50
6.4.3 Holonomy Components 50
6.5 Visualization 51
6.6 Summary 52

7 Constructing Holonomy Components 54
7.1 The Problem 54
7.1.1 Examples 55
7.2 Word Based Construction Method 55
7.2.1 Cycles in Automata 55
7.2.2 Graphically Cycle-Free Automata 56
7.2.3 Algebraically Cycle-Free Automata 57
7.2.4 Non-Aperiodic Automata 59
7.2.5 The Algorithm 62
7.2.6 Examples 63
7.3 Dependency Function Based 64
7.3.1 The Algorithm 67
7.3.2 Example 68
7.4 Summary 71

8 Applications 72
8.1 Understanding the Holonomy Decomposition 73
8.1.1 Decompositions Without any Hierarchical Dependence 73
8.1.2 The Role of Subduction Equivalence Classes 73
8.2 Properties of the Holonomy Decomposition 76
8.2.1 Number of Hierarchical Levels 76
8.2.2 Size of a Component's State Set 77
8.3 Decomposition of the Rings of Integers Modulo n 79
8.3.1 Representation 79
8.3.2 The Extended Set of Images 79
8.3.3 Subduction, Equivalence Relation, and the Tiling Picture 80

8.3.4 Number of Levels 82
8.3.5 Number of States 82
8.3.6 Holonomy Group Components 83
8.3.7 The Lesson 85
8.4 Formal Models of Understanding 85
8.4.1 Representations in Artificial Intelligence 86
8.4.2 The 'What to do?' Problem 86
8.4.3 Capturing Learning 91
8.5 Summary 92

9 Achievements and Future work **93**
9.1 Contributions to Knowledge 93
9.2 Possible Future Research Directions 94
9.3 Exploring a New Landscape 95

A Decompositions of Finite Residue Class Rings of Integers Modulo n, up to $n = 20$ **97**

B Dilworth's Theorem **99**

C Related Publications **100**

D Software Architecture **102**
D.1 `Grasp` 102
D.2 `jGrasp` 102

E Glossary of Symbols **104**

Chapter 1

Introduction

For any finite system a working hierarchical model can be generated automatically.

If we would like to summarize the main theme of this work very shortly, then we could say that it investigates what the last issue of the previous statement mentions: here we give actual algorithms for generating hierarchical models, and once we have the computational tools, we generate models for some particularly interesting finite systems. The statement above has been known to be true for a long time [KR65], so our task is only to start using this result. But this is still a long story.

1.1 The Concept

1.1.1 Models

"The best material model of a cat is another, or preferably the same, cat."
A. Rosenblueth [with Norbert Wiener], Philosophy of Science 1945.

What is a *model of something*? Abstractly speaking, the model is any system which is not thing itself, but it shows some relevant features of the thing or phenomenon to be modeled. In some respect the model should be easier to handle otherwise the thing could be its own model, (and a map with scale 1 : 1 is pretty useless). By a working model we mean that not just the static structure of the original phenomenon is captured, but the model can contain processes as well. Building a model of a system usually involves the identification of its subsystems and their relations. Therefore, the problem of decomposition naturally arises here.

1.1.2 Decompositions

Despite some relatively new scientific approaches[1], scientific understanding still proceeds by taking apart things, identifying their components. Molecules are built up from atoms, atoms from elementary particles, a human brain contains billions of neuron cells, a piece of software is written as code lines of many instructions, etc. By listing the components, we can get to know what are the ingredients for building a given system. Therefore the usefulness of a decomposition method does not need any other justification. But as we go further during a scientific research, the next and more important question is that how those components are put together, how the subsystems are related to each other. We will show that the actual way of wiring the components together is more interesting than the list of the components, and this fact is often neglected. Here we emphasize the usefulness of hierarchical compositions, but this needs some justification.

1.1.3 Hierarchies

We comprehend the world around us in terms of hierarchical representations. We recognize social relations, the structure of organizations by the ranking of the members according to some order [Sim96]. We study the physical world along the spatial and size hierarchies from string theory to the galaxies. Software development methodologies like object-orientation structure for computer programs use the hierarchies of both data and procedures [Boo91]. Our decimal base number notation system is also inherently hierarchical. Even the current debate on the definition of emergence revolves around the notion of hierarchical levels. It is beyond question that among our cognitive models hierarchies are pervasive. One might say that this is a constraint on our cognition albeit a very fruitful one, thanks to the nice properties of hierarchies:

- information flow between levels are restricted enabling modularity (also within one level with parallel components).
- generalization and specialization are natural operations realized by taking subsets of levels in either direction up or down the hierarchy.

It's not the case that all systems are hierarchical, rather the opposite is true. Natural systems have *tangled hierarchies*, hierarchies with *strange loops*: "The Strange Loop phenomenon occurs whenever, by moving upwards (or downwards) through the levels of some hierarchical system, we unexpectedly find ourselves right back where we started" [Hof79]. But even

[1] A nice example is the notion of emergence, where the system is understood by describing simple low-level rules that spontaneously lead to complex behavior [Joh01].

in those cases in understanding them we first use a hierarchical way then we introduce the strange loops as a deviation from hierarchies.

We will not consider the philosophical question here, whether hierarchies are out there in reality or are they just guiding principles of our minds when comprehending the world around us. For the sake of simplicity here it is enough to assume the latter proposition only.

1.1.4 Coordinate Systems

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 original phenomenon. An obvious example is the Descartes coordinate system, where we can uniquely specify any point of the n-dimensional space by n coordinates. However, this is an example of an inherently non-hierarchical coordinate system for a totally homogenous system. In general different coordinates have different roles, addressing parts of the system different in size, function, etc. The natural example of a hierarchical coordinate system is our decimal positional number notation system: different coordinates correspond to different magnitudes.

Here we consider coordinate systems that are hierarchical and algebraically produced for and from a finite state automata.

1.1.5 Aspects Do Matter

Things can be described in several forms. Each form represents the same structure but from a different viewpoint and from each distinct viewpoint something else can be seen. Just as walking around a building may support a deeper understanding of it. What is the building for? How big is it? How many people are in there? Examining several sides may shed light even on the inner structure. It might happen that we do not gain any new information from a different aspect so the different point of views vary in their usability in this respect. Moreover, for different purposes they have different values. For example you can enter the building on the front but not on the rear side. Using different approaches, evaluating them on the base of current purposes, motivations, switching between them – these are probably deeply in our cognitive structure.

It is beyond question that in mathematics these techniques are basic. Given a mathematical structure which is hard to study but it can be mapped to an another domain of well-known constructions, this way the problem is almost solved.

1.2 The Substrate

1.2.1 Automata

Regarding the nature of the models discussed here we have a restriction: they should be described as finite state automata. We claim that this is not really a constraint. Automaton is a concept which is general enough to grasp many interesting phenomena of the world around us. Anything which has states and changes its current state responding to external input can be considered as an automaton.

Many phenomena fit into this scheme: organisms responding quickly to the changes of the environment, chemical reactions, many aspects of real computers, especially language processing, and so on. The wide applicability is due to the very strong abstraction which focuses on the very important notion of change [Ash56].

1.2.2 Semigroups

Groups are mathematical structures capturing the notion of symmetry (reversible processes, or operations that can be undone). Algebraically semigroups are the generalization of the concept of the group. Semigroups can capture irreversibility as well, not just symmetry like groups, i.e. there can be operations that cannot be undone. The only one requirement is that the passage of time should be preserved by the associativity of the operation.

In this work the role of the semigroups is that they are the algebraic aspects of automata: the input symbols of an automaton can be considered as transformations of the state set, as functions. Therefore we can use the precise mathematical tools available in algebra for studying the phenomena being described as an automaton. Philosophically, this aspect also gives us a very nice level of abstraction for computational structures: we can consider processors and memory as the very same resource, since they are not distinguished in the semigroup.

1.2.3 Emulation

It should not be surprising that in general, when we decompose something, i.e. identify its components and determine the rules how to put them together, finally we do not get back exactly the same thing. If it is “smaller”, then we got the decomposition wrong (or we can say that we have an approximation), if it is “bigger” in some sense, then we talk about emulation.

Emulation is an easy concept in computer science: a machine $\mathcal{A}_2$ emulates $\mathcal{A}_1$ if $\mathcal{A}_2$ can do everything what $\mathcal{A}_1$ can do. It might be able to do more, but we should be able to use $\mathcal{A}_2$ instead of $\mathcal{A}_1$ in any case.

Clearly, it is an important issue how to interpret certain operations of $\mathcal{A}_2$ as the operations of $\mathcal{A}_1$. We will consider this in full detail. In alge-

braic terms, this will be done by the mappings of the homomorphism from subautomaton for establishing the division (emulation), see Section 2.1.5.

1.2.4 Finiteness, Computational Complexity

Here we deal only with finite structures, but this does not mean that the theory is unable to deal with infinite structures (e.g. [Neh92]). We have the restriction, since we focus on computational implementations and applications of the theory.

Our computers are finite, plants, animals are finite and we humans are also finite in time and in space as well. The actual loss we have is the pathology of infinite machines only (many uncomputable functions and undecidable problems). Of course the argument is right that a Turing-machine has more computational power than a finite-state machine or a stack-machine. But let's consider the case of palindromes. In reality we do not have to deal with palindromes with arbitrary length, and if words to be checked are finite then we can tackle the recognition problem with a finite-state machine.

If we stay within the finite realm, we still have serious difficulties. Calculating a decomposition is not a simple task, and at every stage of the algorithm, combinatorial explosions may come up. But there is some hope due to the following considerations:

- For practical purposes we need a reasonably good decomposition, not the most optimal (e.g. the shortest possible) one.
- There might be domains of interesting special problems which admit an efficiently calculable decomposition.
- We can use only approximations (not fully calculating all the hierarchical levels).

Unfortunately we cannot entirely get rid of undecidable problems even in the finite case. For example the potential divisibility in finite semigroups is undecidable [KS98].

1.3 Research Questions and Motivations

1.3.1 Feasibility

Our very first question is more than obvious: Is it really possible? Can we calculate such decompositions? Clearly, in the 60's the available computational power of computers was not enough for a challenge like this. But today's computers are more powerful, and the software development tools and computer algebra systems give a lot of help in attacking difficult problems. At least it is time to try.

There is another issue which makes this work timely. Being a mathematician and being a computer scientist (or a programmer), though they are quite close to each other, still require different mind sets. Proving that a given mathematical object exists concludes the work of the mathematician, but that is exactly the beginning of the work of the programmers, since that given element should be found possibly in a very huge set.

So our first research issue stated precisely is the following:

0. It is necessary to investigate the computational feasibility of the decomposition algorithms in order to develop the required software. This requires the comparison and evaluation of different decomposition methods.

The ordinal number 0 here emphasizes that the computational tool for decompositions is not the end product of this work, rather a prerequisite for the actual research.

1.3.2 Exploration and Exploitation

Once we have the computational tool, we can analyse such decompositions that are otherwise not available by manual calculations. Thus, we can start a systematical exploration of specific classes of finite state automata.

1. Study interesting examples for gaining knowledge about the nature of automatically generated decompositions.

The improvement of the decomposition algorithms also remains in focus, since we assume that the more we know about an algorithm the better we can perform.

2. How can we use theoretical insights gained in the exploration phase for improving the decomposition methods?

1.3.3 Formal Models of Understanding

We mentioned before that a cascaded decomposition can be considered as a coordinate system for understanding a given phenomenon. Possible applications of this idea pop up in all different fields where we deal with hierarchical models of systems: physics, where the top level[2] coordinates can be considered as conserved quantities of the system, while the symmetries comprise the bottom level [Rho71]. In software-development the formal models of understanding might provide tools for automated programming, since developing a piece of software is just creating a sophisticated cognitive model

[2]There is an ambiguity between the different meanings of top and bottom level, here we refer the most independent level as top.

[Neh94]. In artificial intelligence [Neh96a] embodied agents equiped with the ability of creating formal models from data coming from their sensors could change their representation of the environment on the fly having a great advantage over purely reactive agents or agents with fixed representations. Recently biological sciences produce a huge amount of data that remain largely uninterpreted so far. As a prominent example one might mention the hype around the sequencing of the human genome, which, while undoubtly representing great progress, comprises a big finite description that we only partially understand. In evolutionary biology it is still contentious how complexity changes in the course of evolution. This is due to the unclarified notions of complexity, which can be clearly defined in terms of hierarchical decompositions [NR00].

This short summary of possible applications shows that this research is not only motivated but that even the prospects for the near future results could potentially be highly rewarding. Therefore our final research question, which is only partially answered here is the following.

3. How can we use a cascaded decomposition as a coordinate system providing a formal model of understanding?

1.4 Roadmap

The first chapter introduced the fundamental notions needed for understanding this research. The definitions here are very informal, they stay at a very abstract, almost philosophical level, but they should help in understanding the details of what follows.

Chapter 2 presents the mathematical background and fixes the notation used in subsequent chapters.

Chapter 3 briefly introduces the Krohn-Rhodes Prime Decomposition Theorem, which is the basis of this work.

Chapter 4 describes an iterative proof technique for the Krohn-Rhodes Theorem, the $V \cup T$ technique, and discusses its applicability for practical problems.

Chapter 5 contains the full proof of the Holonomy Decomposition Theorem.

Chapter 6 discusses the general details of a computational implementation for the holonomy decomposition.

Chapter 7 deals with the main problem of constructing the holonomy components. It describes two different methods for solving the problem.

Chapter 8 shows some preliminary applications of the computational tool.

Chapter 9 summarizes the achievements and delineates the possible directions for future research.

Wherever it makes sense, there is a section with illustrative examples.

Chapter 2

Mathematical Preliminaries and Notations

"Let no one unversed in geometry enter here."
Written over the gate of Plato's Academy.

Here we establish the close connection between finite state automata and semigroups. The related notions, division and emulation, wreath and cascaded product, etc., show that automata and transformations are just the two different sides of the same coin. For the sake of brevity, only those notions are defined which are needed for the proofs in this work, for more details see [DN05, Arb68, Eil76].

The notation applied here is slightly different compared to previous works. We tried to change it for the better, to promote understanding. We use lowercase letters for elements of sets, capital letters for sets, and calligraphic letters for sets of sets or for relations.

We denote the set of integers $\{0, 1, \ldots, n-1\}$ by $\mathbf{n}$.

2.1 Semigroups and Groups

2.1.1 Semigroups

A *semigroup* is a set S equipped with an associative binary operation $\mu : S \times S \to S$. Instead of $\mu(s_1, s_2)$ we write $s_1 \cdot s_2$ or more briefly $s_1 s_2$. If A and B are subsets of a semigroup, then AB means the set $\{ab : a \in A, b \in B\}$. An element 1 is the identity element of S if $s1 = 1s = s$, $\forall s \in S$. The identity is unique if it exists. By S^1 we denote S if it has an identity otherwise $S \cup \{1\}$. By S^I we mean $S \cup \{I\}$ where I is a new element that acts as an identity on S and itself, the identity of S (if it exists) ceases to be an identity as it fails on I. An element $r \in S$ is called a *right-zero element* of S if $sr = r$, for all $s \in S$. Symmetrically, $\ell \in S$ is a *left-zero element* if $\ell s = \ell$, for all $s \in S$. In addition, $o \in S$ is the *zero element* if $os = so = o$, $\forall s \in S$. The zero element

is also unique if it exists. The *order* of a semigroup S is its cardinality $|S|$. We say that a subset A of S generates the semigroup $\langle A \rangle = S$ if all elements of S can be expressed as a finite product of elements in A. A semigroup S is *aperiodic* if for each element $s \in S$ there is a positive natural number n such that $s^n = s^{n+1}$; for a finite semigroup this means that it contains no nontrivial subgroups.

2.1.2 Groups

A semigroup is a *monoid* if it has an identity element. A monoid is a *group* if for every $s \in S$ there is an inverse $s^{-1} \in S$ such that $ss^{-1} = s^{-1}s = 1$. A subset T of a semigroup S is a *subsemigroup* if it is closed under the multiplication of S. Subgroups are defined analogously. A subgroup H of a group G is *normal* if $gH = Hg\ \forall g \in G$. A nontrivial group is *simple* if it has no nontrivial proper normal subgroups.

Another definition of aperiodicity can be given by using subgroups: A finite semigroup each of whose subgroups has only one element is called aperiodic.

We denote the one element trivial group simply by the identity 1, and if it is a trivial permutation group (see below) then we also indicate the number of states it acts on: 1_n, i.e $(\mathbf{n}, 1)$. C_n is the cyclic group of order n, S_n is the symmetric group on n points. D_n is the dihedral group of order n. $G_{n:k}$ denotes a semidirect product $C_n \rtimes C_k$. G_n is a group with order n with trivial Frattini subgroup[1], where the Frattini subgroup is the intersection of maximal subgroups (or equivalently, the subgroup of non-generator elements).

2.1.3 Transformations and Permutations

In algebraic automata theory we often use the following representation of abstract semigroups and groups.

For a nonvoid finite set A, a mapping $\varphi : A \to A$ is called a *transformation* of A. We denote the identity transformation by 1_A. Instead of $\begin{pmatrix} 1 & 2 & \dots & n \\ i_1 & i_2 & \dots & i_n \end{pmatrix}$ we use a simpler notation $(i_1 i_2 \dots i_n)$, which is not to be confused with group theoretical cyclic notation. If the mapping is bijective, then it is a *permutation.* The *image* of φ is defined as $\{a\varphi : a \in A\}$ denoted by $\mathrm{im}(\varphi)$. If the image of a mapping is a singleton then the mapping is *constant.* The *rank* of a transformation is the cardinality of its image. The set T of all transformations of A form a semigroup under the operation of function composition of transformations and it is called the *full transformation semigroup* denoted by $\mathcal{T}_A = (A, T)$. If S is a subsemigroup of T then (A, S) is called a *transformation semigroup* on A (or briefly a *ts*), and we

[1]For identifying certain groups in our automated decompositions we used the Small Groups data library for `GAP`[gap02].

say that S *acts* on A. There is a subtle issue regarding (A, S^I): S might be a monoid already, but the identity element might not be the identity map on A, therefore in the case of transformation semigroups we add the identity transformation as the new identity element, $(A, S^I) = (A, S \cup \{1_A\})$.

(A, S) is a *permutation group* if each element $s \in S$ acts on A by permutation. We write $a \cdot s$ for the image of state a under the transformation s, and we have $(a \cdot s_1) \cdot s_2 = a \cdot (s_1 s_2)$ for all $a \in A, s_1, s_2 \in S$. It is a basic fact of semigroup theory that every finite semigroup can be represented as a ts using the *right regular representation* (S^1, S) where S acts on S^1 by multiplication on the right [CP67]. If (A, S) is a transformation semigroup, we denote by $(A, \overline{S})$ the transformation semigroup with transformations $\overline{S} = \{t \mid t \in S \text{ or } t \text{ is constant}\}$.

For the canonical state set, we use the notation $\mathbf{n}$ for n points $\{0, \ldots, n-1\}$.

2.1.4 Green's Relations

A subsemigroup I of S is called an *ideal* if $SIS \subseteq I$, and a *left ideal* if $SI \subseteq I$. If $s \in S$ then $S^1 s S^1$ is the *principal ideal* of s, and $S^1 s$ is the *left principal ideal* of s. *Right ideals* and *right principal ideals* are defined analogously.

Analogous to divisibility the Green's relations $\mathcal{L}, \mathcal{R}$ and $\mathcal{J}$ are defined as follows: For $s_1, s_2 \in S$, $s_1 \leq_{\mathcal{L}} s_2$ if $S^1 s_1 \subseteq S^1 s_2$, or equivalently (emphasizing the similarity to divisibility) if there exists some $x \in S^1$ such that $s_1 = x s_2$. This comprises a transitive relation. If $s_1 \leq_{\mathcal{L}} s_2$ and $s_2 \leq_{\mathcal{L}} s_1$ then we write $s_1 \mathcal{L} s_2$, i.e. they generate the same left principal ideals, and we say that s_1 and s_2 are $\mathcal{L}$-equivalent. $\mathcal{R}$-equivalence is defined dually. Similarly,$s_1 \leq_{\mathcal{J}} s_2$, if $S^1 s_1 S^1 \subseteq S^1 s_2 S^1$, or equivalently if there exist some $x, y \in S^1$ such that $s_1 = x s_2 y$. Thus, two elements of a semigroup are $\mathcal{J}$-equivalent if they generate the same principal ideals. The $\mathcal{J}$-equivalence class of $s \in S$ is denoted by $J(s)$ (similarly $L(s), R(s)$).

2.1.5 Homomorphisms

Let S and T be semigroups with multiplications $\circ, \diamond$ respectively and having a mapping $\psi : S \to T$ such that $\psi(s_1 \circ s_2) = \psi(s_1) \diamond \psi(s_2)$, for all $s_1, s_2 \in S$. Then we say that ψ is a *homomorphism* from S to T, a mapping which preserves products. If a homomorphism is bijective then it is an *isomorphism*.

Another definition of simple groups can be given by using homomorphisms: a nontrivial group is simple if its homomorphic images are just itself and the one element group (up to isomorphism).

Division

We say that a transformation semigroup (A, S) *divides* (B, T) denoted by $(A, S) \mid (B, T)$ if we can choose for each $a \in A$ at least one $\tilde{a} \in B$ as a *lift* and and for each $s \in S$ at least one $\tilde{s} \in T$ as a *lift*, such that the following conditions hold. We denote the set of lifts of a state a by $\Lambda(a)$ (and $\Lambda(s)$ for a transformation s respectively).

1. Each member of B (resp. T) is a lift of at most one element of A (resp. S), i.e. the (non-empty) lift sets for distinct elements are non-intersecting. Formally: $\Lambda(a) \neq \emptyset$, $\Lambda(s) \neq \emptyset$, and $\Lambda(x) \cap \Lambda(y) \neq \emptyset \implies x = y$.
2. If $\tilde{a}$ is any lift of a and $\tilde{s}$ is any lift of s, then $\tilde{a} \cdot \tilde{s}$ is some lift of $a \cdot s$, i.e. the products are respected.

Note that in general $\Lambda(a) \cdot \Lambda(s) \subseteq \Lambda(a \cdot s)$, instead of being equal.

$\Lambda(a) \cdot \Lambda(s) \subseteq \Lambda(a \cdot s)$ action in (B, T)

$a \cdot s = a \cdot s$ action in (A, S)

Lift sets for states and transformations might have their internal structure which do not play any role in the division. Moreover the union of lift sets might be a proper subset of B or T. Thus (B, T) is "bigger, richer in structure, can do more", therefore we also say (B, T) *covers* or *emulates* (A, S). In practice, to establish the division it is enough to lift the states and a generator set for the semigroup and check $\Lambda(a) \cdot \Lambda(s_1) \cdots \Lambda(s_n) \subseteq \Lambda(a \cdot s_1 \cdots s_n)$ for all $n \geq 1, \quad s_i \in G, \quad a \in A$ where G is a generator set for S [DN05].

2.1.6 Words and the Free Semigroup.

Let X a set of letters be called the *alphabet*. A *word* over the alphabet X is a finite sequence of elements of X: $(x_1, x_2, \ldots, x_n)$, $x_i \in X$. The empty word is denoted by λ. X^+ is the set of all non-empty finite words. X^+ is a semigroup under the operation of concatenation, it is called the *free semigroup* on X. $X^* = X^+ \cup \{\lambda\}$ is the *free monoid* on X.

A word $v \in X^*$ is a *factor* of a word $z \in X^*$ if there exist words $u, w \in X^*$ such that $z = uvw$. v is a *left factor* of z if there exists a word $w \in X^*$ such that $z = vw$. A word w is *primitive* if it is not a power of another word. For any nonempty word w, the smallest factor u such that $w = u^n$, $n \geq 1$ is the *primitive root* of w. We also use the notation $u = \sqrt{w}$.

Standard references are [Shy01] and [Lot83].

2.2 Finite State Automata

By a finite state *automaton*, we mean a triple $\mathcal{A} = (A, X, \delta)$ where A is the (finite nonempty) *state set*, X is the *input alphabet* and $\delta : A \times X \to A$ is the *transition function.* We do not explicitly consider the output of the automaton as it can be recovered from the state and the input symbol. We tacitly use the state as the output.

We can naturally extend the transition function for words i.e. sequences of input symbols: for the empty word $\delta(a, \lambda) = a$, and for arbitrary words $u, v \in X^*$, $\delta(a, uv) = \delta(\delta(a, u), v)$. There is a natural equivalence relation, the *congruence induced by* $\mathcal{A}$ on words $u \equiv v$ if $\delta(a, u) = \delta(a, v)\ \forall a \in A$, i.e. identifying words with the same action on A. The *characteristic semigroup* $S(\mathcal{A})$, also called the *semigroup of the automaton*, is the set equivalence classes $X^+/\equiv$ of this congruence, with associative operation induced by concatenation. With the characteristic semigroup we can handle an automaton $\mathcal{A}$ as a transformation semigroup $(A, S(\mathcal{A}))$. Conversely if S is a semigroup then the corresponding automaton is $\mathcal{A}_S = (S^1, S)$, where the transition function is the right action of S on S^1. Clearly, $S(\mathcal{A}_S) \cong S$.

An automaton $\mathcal{A}$ *emulates* another one $\mathcal{B}$ with states B if every computation which can be done in $\mathcal{B}$ can be done in $\mathcal{A}$ as well, i.e. $(B, S(\mathcal{B}))$ divides $(A, S(\mathcal{A}))$.

Using automata terminology constant mappings in transformation semigroups are often called *resets.* A *permutation-reset* automaton is an automaton such that each of its inputs acts either as a permutation or a constant map on states.

The *state transition graph* $D(\mathcal{A})$ of an automaton $\mathcal{A} = (A, X, \delta)$ is a digraph with A as the set of vertices and (a, x, b) is a labeled edge if $a \cdot x = b$, where $a, b \in A,\ x \in X$. It is a *loop-edge* if $a = b$. A *path* is a sequence of edges (a_i, x_i, b_i) $1 \leq i \leq n$ with $a_{i+1} = b_i$ for all $1 \leq i < n$, and the *label* of the path is $x_1 \dots x_n$. A *loop* is a path with $b_n = a_1$.

2.3 Wreath Product

Although the concept of the wreath product is not so complicated, it is not as easy to present the intuitive idea how the loop-free cascaded product works. After reading the formal definition a figure may shed light on how state transitions happen in the product (Fig. 2.2). It is also a great help first to consider a simpler product with no dependence between the components.

Let $(A_n, S_n), \dots, (A_1, S_1)$ be transformation semigroups called *components.* The indices $1, \dots, n$ are called *coordinates.* The *direct product* $(A_n, S_n) \times \dots \times (A_1, S_1)$ is the ts $(A_n \times \dots \times A_1, S_n \times \dots \times S_1)$ with the componentwise action

$$(a_n, \dots, a_1) \cdot (s_n, \dots, s_1) = (a_n \cdot s_n, \dots, a_1 \cdot s_1).$$

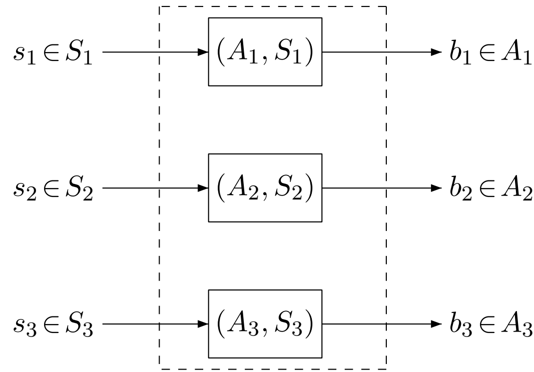


Figure 2.1: State transition in the direct product $(A_3, S_3) \times (A_2, S_2) \times (A_1, S_1)$. The transformation (s_3, s_2, s_1) is applied to state (a_3, a_2, a_1) yielding $(b_3, b_2, b_1) = (a_3 \cdot s_3, a_2 \cdot s_2, a_1 \cdot s_1)$. We use the state as the output of the automaton.

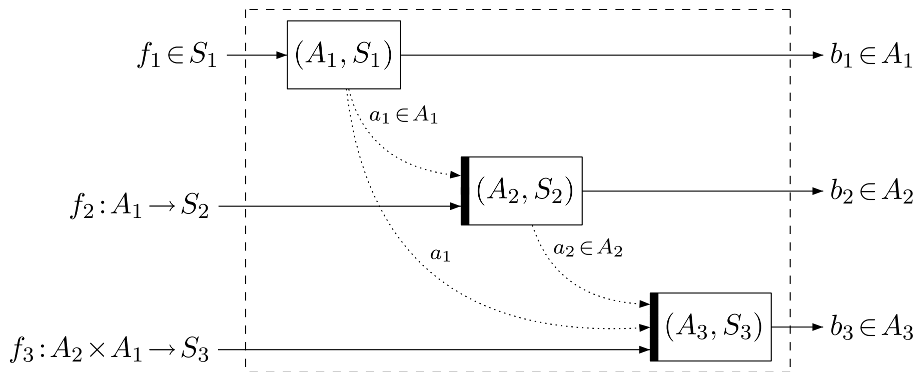


Figure 2.2: State transition in the wreath product $(A_3, S_3) \wr (A_2, S_2) \wr (A_1, S_1)$. The transformation (f_3, f_2, f_1) is applied to state (a_3, a_2, a_1) yielding $(b_3, b_2, b_1) = (a_3 \cdot f_3(a_2, a_1), a_2 \cdot f_2(a_1), a_1 \cdot f_1)$. The black bars denote the applications of functions f_2, f_3 according to hierarchical dependence. Note that the applications of these functions happen exactly at the same moment since their arguments are the previous states of other components, therefore there is no need to wait for the other components to calculate the new states. We use the state as the output of the automaton.

Direct product is also called *parallel composition* as the components' state transitions do not depend on each other, and the order of the components does not really matter up to isomorphism (Fig. 2.1).

Now we introduce an order-dependent connection between the components. Let $A = A_n \times \ldots \times A_1$ and $\mathcal{T}_A$ the full ts on A. Let S be the subsemigroup of $\mathcal{T}_A$ consisting of all transformations $s : A \to A$ satisfying the condition of *hierarchical dependence* of coordinates: Denote $p_k : A \to A_k$ the kth projection map, then for each $k = 1, \ldots, n$ there exists $f_k : A_{k-1} \times \cdots \times A_1 \to S_k$ such that

$$p_k\big((a_n, \ldots, a_{k+1}, a_k, \ldots, a_1) \cdot s\big) = a_k \cdot f_k(a_{k-1}, \ldots, a_1) = a'_k$$

where $s \in S,\ a_k, a'_k \in A_k,\ k = 1, \ldots, n$. That is, the new kth coordinate a'_k resulting from the action of s depends only on the values of the old first k coordinates and on the transformation s. Moreover, it is given by acting with an element of S_k which depends only on s and $(a_{k-1}, \ldots, a_1)$. We can write this transformation as the ordered list of these functions: $s = (f_n, \ldots, f_1)$. f_i gives the component action in the ith position. We call these functions *dependency functions.*

Then the transformation semigroup $(A, S) = (A_n, S_n) \wr \cdots \wr (A_1, S_1)$ is the *wreath product* of transformation semigroups $(A_n, S_n), \ldots, (A_1, S_1)$. Reading from left to right the last component is the top level of the hierarchy.

The multiplication in the wreath product is carried out by concatenating functions. Let $s = (f_n, \ldots, f_1)$ and $t = (g_n, \ldots, g_1)$ elements of S and for the sake of brevity, where the arguments of the functions are straightforward, they are not displayed, e.g. $f_i()$ means $f_i(a_{i-1}, \ldots, a_1)$. Then $s \cdot t = (m_n, \ldots, m_1)$ can be given by:

$$m_1 = f_1 \cdot g_1 \tag{2.1}$$

since they are elements of the semigroup S_1 it is normal semigroup multiplication. However for lower levels it is more complicated and can be given in respect a particular state $(a_n, \ldots, a_1)$:

$$m_i = f_i() \cdot g_i\big(a_{i-1} \cdot f_{i-1}(), a_{i-2} \cdot f_{i-2}(), \ldots, a_1 \cdot f_1\big), \tag{2.2}$$

and clearly m_i is again a function of $(a_{i-1}, \ldots, a_1)$ to S_i. If we write $(a'_n, \ldots, a'_1)$ for $(a_n, \ldots, a_1) \cdot s$ then the equation can be abbreviated to $m_i = f_i() \cdot g_i(a'_{i-1}, \ldots, a'_1)$.

We also use the notation f_i^s for a dependency function, where i indicates the hierarchical level as above, and s is a given cascaded transformation just to make it clear where the function belongs to.

By a *cascaded state* we mean a tuple of component states as above, and by a *cascaded action* we mean an actual tuple of component actions (this is not to be confused with the cascaded transformation, which is a tuple of dependency functions).

2.4 Examples

2.4.1 Faces of an Automaton

Here we show a very simple automaton in several different ways to emphasize the fact that though being the same thing, it does matter in which form of the automaton we try to work with.

Example 2.1 Let $X = \{x, y\}$, $A = \{a, b, c\}$, and δ as following:

$$\begin{array}{ll} \delta(a,x) = a, & \delta(a,y) = b, \\ \delta(b,x) = c, & \delta(b,y) = b, \\ \delta(c,x) = a, & \delta(c,y) = b. \end{array}$$

This is a description of a very simple machine. It's not hard to find out what it does but other forms of the machine are more easily understandable.

Transition Table

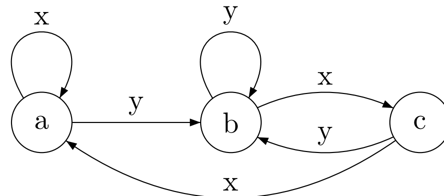
Transition tables describe a machine by giving the values of the transition function for each state-input pair. It is basically a shorthand notation for defining the δ function.

		input	
		x	y
state	a	a	b
	b	c	b
	c	a	b

It's easy to comprehend for humans when the size of the table is relatively small. For larger machines it still helps in tracking down the state transitions for a given input sequence. It also gives a straightforward data structure for representing abstract machines in a computer program.

Diagram

For human perception and comprehension the most suitable representation is visual. The diagram form can be considered as a flowchart or illustration of the inner workings of the machine, actually the algorithm which it implements.



Matrix

The machine action is described by boolean matrices and a specific state is represented by a vector. For each input symbol there is a matrix which has one row and column for each machine state. The i, j entry is 1 if the corresponding input symbol causes a state transition from state i to j, otherwise it is 0. States are represented as vectors.

$$\begin{array}{cc} \begin{array}{c c} & \begin{array}{ccc} a & b & c \end{array} \\ \begin{array}{c} a \\ b \\ c \end{array} & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \end{array} & \begin{array}{c c} & \begin{array}{ccc} a & b & c \end{array} \\ \begin{array}{c} a \\ b \\ c \end{array} & \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} \end{array} \\ x\text{-matrix} & y\text{-matrix} \end{array}$$

$$a = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \quad b = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \quad c = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$$

The state transitions caused by an input sequence starting from a specified initial state can be found by just multiplying the corresponding state vector on the right with the matrices according to the input sequence.

Regular expression

If we consider a machine as a tool for recognizing a language then the most useful notation is a regular expression. In our example, if we have the initial state a and the accepting state c, then the machine accepts all words of x's and y's that end in yx. So the accepted language is:

$$L = \{y, x\}^* \{yx\}$$

Function

Let X and A sets. Then a machine is a function $f : X^* \to A$, i.e. mapping the sequences of input symbols to states (which can be considered as outputs).

Semigroup

The characteristic semigroup of our automaton consists of 4 elements corresponding to input sequences: $x = [1, 3, 1], y = [2, 2, 2], z = [3, 3, 3], v = [1, 1, 1]$, where z corresponds to the input sequence yx and v to yxx. The operation is given by the following multiplication table:

	x	y	z	v
x	v	y	z	v
y	z	y	z	v
z	v	y	z	v
v	v	y	z	v

2.4.2 Building a Modulo 4 Counter Hierarchically

Since wreath products are usually far too complex objects to describe them in full detail, we use a very small example for demonstrating the cascaded composition.

Example 2.2 *We would like to build a modulo 4 counter as a wreath product of two modulo 2 counters. By a counter modulo n we mean the permutation group $C_n = (\boldsymbol{n}, \langle +1 \rangle)$, where $\boldsymbol{n} = \{0, 1, \ldots, n-1\}$. We would like to build*

$$C_4 \mid C_2 \wr C_2.$$

The state set of the $C_2 \wr C_2$ is $\mathbf{2} \times \mathbf{2} = \{(0,0),(0,1),(1,0),(1,1)\}$, which is basically the binary representation of the integers from 0 to 3. For the transformation we need to lift $\{+1,+2,+3,+4\}$, by describing them as a 2-tuple of dependency functions.
We can give a lift easily for the operation of incrementing by 1. On the top (least dependent) level it is always +1, on the next level the action depends on whether we have a carry or not.

$\widehat{+1}$:

$$\begin{aligned} f_1^{+1}() &= +1 \\ f_2^{+1}(0) &= i \\ f_2^{+1}(1) &= +1 \end{aligned}$$

where $i = +1 \cdot +1$ is the identity. Now we calculate the lift of +2 as $\widehat{+1} \cdot \widehat{+1}$ according to equations 2.1 and 2.2.

$\widehat{+2}$:

$$\begin{aligned} f_1^{+2}() &= f_1^{+1}() \cdot f_1^{+1}() = +1 \cdot +1 = i \\ f_2^{+2}(0) &= f_2^{+1}(0) \cdot f_2^{+1}(1) = i \cdot +1 = +1 \\ f_2^{+2}(1) &= f_2^{+1}(1) \cdot f_2^{+1}(0) = +1 \cdot i = +1 \end{aligned}$$

Note, that when defining $f_2^{+2}(0)$ the second factor is $f_2^{+1}(1)$ instead of $f_2^{+1}(0)$, since the state transition by $f_1^{+1}()$ has been made.

$\widehat{+3}$ calculated as $\widehat{+2} \cdot \widehat{+1}$:

$$\begin{aligned} f_1^{+3}() &= f_1^{+2}() \cdot f_1^{+1}() = i \cdot +1 = +1 \\ f_2^{+3}(0) &= f_2^{+2}(0) \cdot f_2^{+1}(0) = +1 \cdot i = +1 \\ f_2^{+3}(1) &= f_2^{+2}(1) \cdot f_2^{+1}(1) = +1 \cdot +1 = i \end{aligned}$$

since $f_1^{+2}() = i$.

$\widehat{+4}$ calculated as $\widehat{+3} \cdot \widehat{+1}$:

$$\begin{aligned} f_1^{+4}() &= f_1^{+3}() \cdot f_1^{+1}() = +1 \cdot +1 = i \\ f_2^{+4}(0) &= f_2^{+3}(0) \cdot f_2^{+1}(1) = +1 \cdot +1 = i \\ f_2^{+4}(1) &= f_2^{+3}(1) \cdot f_2^{+1}(0) = i \cdot i = i \end{aligned}$$

and it is the identity of the cascaded product, as expected: $\widehat{+4} = \widehat{i}$.

Now let's see how the action works in the wreath product.

$$\widehat{0} \cdot \widehat{+1} = (0,0) \cdot \widehat{+1} = (0 \cdot f_2^{+1}(0), 0 \cdot f_1^{+1}()) = (0 \cdot i, 0 \cdot +1) = (0,1) = \widehat{1}$$

which is $\hat{1}$. Now let's see what happens if we add 3 to 1 in the wreath product:

$$\widehat{1} \cdot \widehat{+3} = (0,1) \cdot \widehat{+3} = (0 \cdot f_2^{+3}(1), 1 \cdot f_1^{+3}()) = (0 \cdot i, 1 \cdot +1) = (0,0) = \widehat{0}.$$

Note that his example is very special in several respects. The components are groups, and we have embedding $(\mathbf{4}, C_4) \hookrightarrow (\mathbf{2}, C_2) \wr (\mathbf{2}, C_2)$, instead of the more general division.

2.5 Summary

We have presented the very basic notions of algebraic automata theory with emphasis on the description of the wreath product from the computer scientist's viewpoint.

Chapter 3

The Krohn-Rhodes Theory

The previous chapter introduced mathematical notions and structures but it did not tell anything explaining why we need them. Now it is time to show the prize for the efforts. We present the Krohn-Rhodes Prime Decomposition Theorem, which is a theorem about the algebraic decomposition of finite state automata. The importance of the theorem described here and its implications should convince the reader that it is worth going on. To shed light on the main ideas here we use a special cognitive tool; we expose the key points with the help of a metaphor.

We restrict considerations from now on to finite automata.

3.1 The Prime Decomposition Metaphor

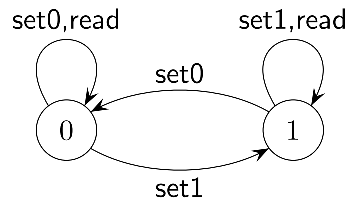
"The essence of metaphor is understanding and experiencing one kind of thing in terms of another." [LJ80] Metaphor can be considered as a cognitive aid, when we understand an unknown thing in terms of a well-known one. Here the familiar thing is the set of integers, and the less understood phenomenon is finite computation, more precisely finite automata. We would like to see the structure of finite computations, how more complicated computations are built from simpler pieces, what are the elementary building blocks like the primes which can not be divided further. For most of the time science is about decomposing, disassembling things and trying to understand how the pieces are put together. In the case of integers the way of putting together numbers is simply multiplication, in the case of automata it is more complicated, we need to use cascaded composition. The basic building blocks of automata are also more complicated, there are two types of them, and they have inner structure as well.

3.2 The Building Blocks

Roughly speaking, we have 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 it is irreversible, since there is no way to restore the previously stored data. Corresponding to these two types we have two kinds of components: *simple group automata* for reversible and the *flip-flop automaton* **F** for the irreversible aspect.

Finite simple group automata are automata whose characteristic semigroups are simple groups. Finite simple groups are now well-described mathematical objects, although they are not as simple as the name suggests since their classification [GLS94] needs long proofs.

The flip-flop automaton **F** can be thought of as a device capable of storing one bit: we have two states $A = \{0, 1\}$ and three symbols in the input alphabet $X = \{\mathsf{set0}, \mathsf{set1}, \mathsf{read}\}$. The read is the identity operation, but since we consider the state as the output of the automaton we can think of it as retrieving what state was set before.



In semigroup theoretic terms it has two resets and one identity, hence its other name is *two-state identity-reset automaton*.

3.3 Wiring the Components

Despite their potential in fostering understanding, every metaphor has its limits. Our prime decomposition metaphor might suggest that the components are the most important and the way they are put together does not really matter. But this is false. Unlike the decomposition of integers, where we use arithmetic multiplication due to commutativity the order of the components is arbitrary, in the case of automata we use the wreath product to put automata together in a hierarchical cascaded way. As we have already seen in Example 2.2, this composition is rather complicated. Even in a very simple case, the explicit description of the dependency functions is very lengthy. On the other hand, as we will see, this is the most interesting part as well.

Neglecting the dependency functions has another reason as well, not just the natural limitation of the decomposition metaphor. One can prove the Krohn-Rhodes Prime Decomposition Theorem without explicitly considering the dependency functions. Shortly, mathematicians do not necessarily need them. They come into focus when we study actual "working" cascaded automata.

3.4 The Krohn-Rhodes Prime Decomposition Theorem

Now we are able to state the main theorem, the basis of this current work.

Theorem 3.1 (Krohn-Rhodes Prime Decomposition Theorem) *Given a finite automaton $\mathcal{A}$, then $\mathcal{A}$ it can be emulated by a cascade product of components from $\{\mathcal{A}_{\mathbf{F}}, \mathcal{A}_{G_1}, \dots, \mathcal{A}_{G_n}\}$, where $\mathbf{F}$ is the flip-flop and G_i, $1 \leq i \leq k$ are simple groups dividing the characteristic semigroup $S(\mathcal{A})$.*

Conversely, let $\mathcal{B} = \mathcal{B}_1 \wr \cdots \wr \mathcal{B}_n$ be a cascade product of automata which emulates the automaton $\mathcal{A}$. If a subsemigroup S of the flip-flop monoid $S(\mathbf{F})$ or a simple group S is a homomorphic image of a subsemigroup of $S(\mathcal{A})$, then S is a homomorphic image of a subsemigroup of $S(\mathcal{B}_t)$ for some component automaton $\mathcal{B}_t$ ($t \in \{1, \dots, n\}$).

The proof here omitted since the next two chapter contains the sketch of one proof and a detailed description of another one.

3.5 Historical Remarks

There are various proofs for the Krohn-Rhodes Theorem, thus a historical summary of their origins might be useful to clarify the situation and to justify the need for refining the proof, namely the work presented here.

The first proof [KR65] was presented in the context of finite state machines which made the argument somewhat difficult to follow. After that new algebraic techniques were introduced. The $V \cup T$-technique [KRT68] uses the Green relations of the semigroup of transformations, but it produces a very long list of components (including repetitions), therefore it cannot be used for practical purposes (see Section 4.2). A more recent version of the $V \cup T$-technique [Neh96b] has partially overcome this problem, but it is still not efficient enough for computational implementations. Zeiger took a different route using covers (more general concept of tiling) [Zei67, Zei68]. Later this approach was called the holonomy decomposition. Zeiger's original proof contained some inaccuracies, and these were corrected in [Gin68]. The weakness of Zeiger's method is in the way of refining covers. Refining

only one equivalence class at once yields unnecessary long list of components. The cure for this is the height function which shows exactly what equivalence classes can be refined in parallel. This was first described by Eilenberg, who made the proof [Eil76] of the holonomy decomposition using partial functions, then Holcombe [Hol82] improved it by identifying cases when between some particular consecutive levels direct product can be used instead of wreath product. Recently, Nehaniv gave a proof [DN05] with a computational implementation in mind. The current proof is the extension and culmination of his work by emphasizing the separated circuitry of the cascaded product and it comes together with working software [ENN03].

Although for practical implementations we usually consider only finite cases, it is worth noting that there are versions of the proof for arbitrary semigroups [HLR88, EN02] (and others). There are also completely different proofs [Ési00, RW89a, RW89b] based on kernels.

3.6 Summary

The Krohn-Rhodes Theory is the basis of this work, and its potential is so high that we will still continue developing and exploiting the implications of the theory.

Chapter 4

The $V \cup T$-technique

Since we have several different proofs for the Krohn-Rhodes Theory, the question naturally arises: which method should be implemented computationally? Due to its simplicity, our first choice is the so called $V \cup T$-technique. This method is one of the earliest proof techniques [KRT68]. It works with semigroups and uses the right regular representation when ts's are needed for the resulting cascaded components.

4.1 The Iterative Construction

The main idea of the algorithm can be summarized in the proof of the following lemma (for the sake of brevity in terms of semigroups). The iterative nature of this lemma gives the working mechanism of the decomposition.

Lemma 4.1 ([KRT68]) *Let S be a finite semigroup. Then either*

(a) *S is left simple, i.e. S is the direct product of a group with A^ℓ, for some set A with multiplication $xy = x$ (the elements are left zeros),*

(b) *S is a finite monogenic semigroup (generated by one element), or*

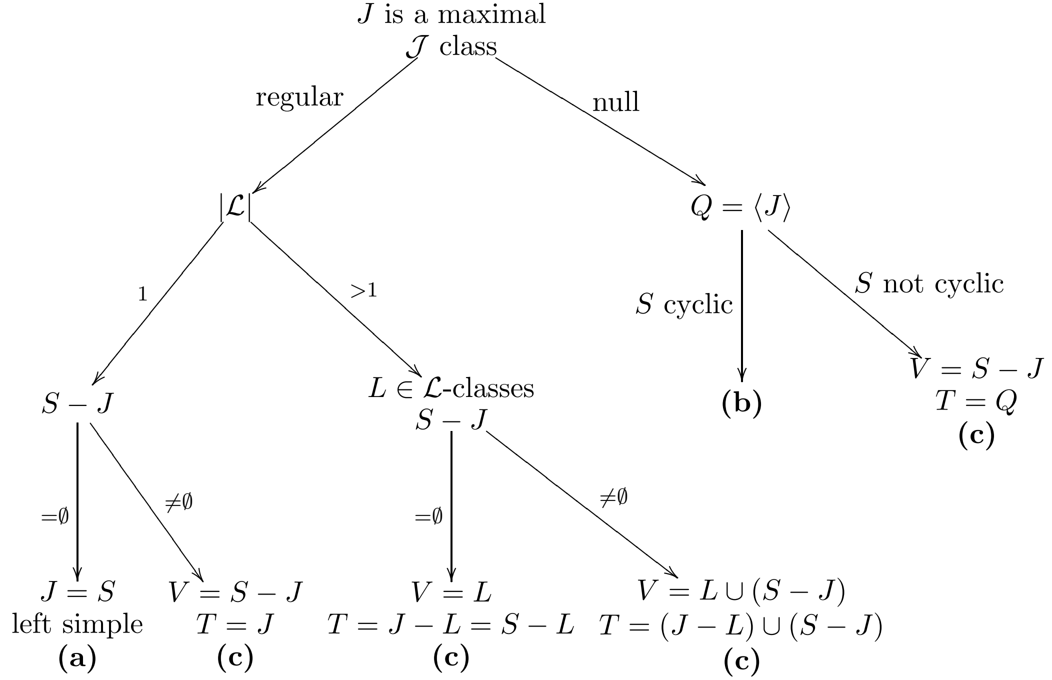
(c) *there exists a proper left ideal $V \subset S$ and a proper subsemigroup $T \subset S$ such that $S = V \cup T$.*

Proof: Let J be a maximal $\mathcal{J}$ class of S. Either J is regular or is a one-point null $\mathcal{J}$ class. Suppose J is regular and has only one $\mathcal{L}$ class. Then J is a subsemigroup of S. Let $F(J)$ be the ideal $S - J$. If $F(J) = \emptyset$, $J = S$ is left simple, case **(a)**. If $F(J) \neq \emptyset$, let $V = F(J)$ and $T = J$, case **(c)**.

Suppose J is regular and has more than one $\mathcal{L}$ class. Let L be one. If $F(J) = \emptyset$, let $V = L$ and $T = J - L = S - L$, case **(c)**. If $F(J) \neq \emptyset$, let $V = L \cup F(J)$ and $T = (J - L) \cup F(J)$, case **(c)**.

If J is not regular, then it is a one-point null $\mathcal{J}$ class. Let $J = \{q\}$, and $Q = \langle q \rangle$. Either $Q = S$, case **(b)**, or let $V = F(J)$ and $T = Q$, case **(c)**. This exhausts the possibilities.

Let's denote $|\mathcal{L}|$ the number of $\mathcal{L}$-classes. Then the proof can be visualized with following diagram. The arrows from a node represent different decisions.



□

The first two cases are easy to decompose into flip-flops and groups but the $V \cup T$ technique stops when finding monogenic or left simple semigroups. A left simple semigroups is a product of a group and a left zero semigroups (every element is a left zero element). A monogenic semigroup divides the direct product of its fuse and its cyclic (simple and abelian) subgroup[1]. For further details see [KRT68].

In the third case we have

$$(S^1, S) \mid (V^I, \overline{V}^I) \wr (T^I, \overline{T}^I).$$

Then we iterate the process by applying the lemma again to V and T (they are both subsemigroups of S) in order expanding the list of components until monogenic or left simple semigroups appear.

[1]This is the usual decomposition of monogenic semigroups. The fuse (or tail) is the aperiodic part.

4.2 Results

4.2.1 The Full Transformation Semigroup

Full transformation semigroups are specially good cases for testing a decomposition algorithm, since regarding their order they are the biggest semigroups on n points. But, we know [Eil76] a nice and compact decomposition for them:

$$\mathcal{T}_n \mid (\mathbf{2}, \overline{S_2}) \wr \ldots \wr (\mathbf{n\text{-}1}, \overline{S_{n-1}}) \wr (\mathbf{n}, \overline{S_n}).$$

So, in the case of $\mathcal{T}$ we expect $\mathcal{T}_2 \mid (\mathbf{2}, \overline{S_2})$. But using $V \cup T$ we get: $\mathcal{T}_2 \mid (\mathbf{1}, \overline{1}) \wr (\mathbf{1}, \overline{1}) \wr (\mathbf{2}, \overline{S_2})$, which seems to be slightly redundant, we have more hierarchical levels than needed. If we act on more points the redundancy becomes worse:

Semigroup	Order	#Hierarchical levels by $V \cup T$
$\mathcal{T}_2$	4	3
$\mathcal{T}_3$	27	19
$\mathcal{T}_4$	256	401

At $\mathcal{T}_4$ the number of hierarchical levels exceeds the order of the semigroup being decomposed. Getting more hierarchical components than n^n for an automaton with n states is far from being efficient. This inefficiency of the $V \cup T$ algorithm originates from the iterative step: V and T may overlap and thus subcomponents may appear again and again. However a variant of the $V \cup T$ proof exists, the $\mathcal{L}^+$-decomposition, which avoids much of the duplications [Neh96b], although not fully alleviating this problem.

4.2.2 More Extreme Examples

The full transformation semigroup might be considered as a special example, since regarding its order it is the biggest semigroup on n states. One might suspect that the length of the decomposition is due to to the symmetric subgroup, but this is not the case. Now we check an aperiodic example.

Example 4.2 *An* elevator *is an automaton with n states (the storeys) and two input symbols u, d (going up and going down) realizing the following transformations:*

$$u(i) = \begin{cases} i+1 & i < n \\ n & i = n \end{cases}$$

$$d(i) = \begin{cases} i-1 & i > 1 \\ 1 & i = 1 \end{cases}$$

The state transition graph basically is a 'line' on which we can move in two directions (see Fig. 4.1). Decomposing elevators and examining the length of the decompositions give the following result:

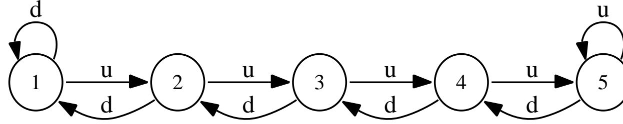


Figure 4.1: An elevator automaton with 5 states.

Number of states	Order	#Hierarchical levels by $V \cup T$
2	2	2
3	7	10
4	17	50
5	34	290

The growth of the number of hierarchical levels is worse than in the case of the full transformation semigroup. Again, most of the components are one element trivial semigroups.

4.3 Summary

The simplicity of the $V \cup T$ method turned out to be deceptive, since the decomposition it provides is unusably complex due to its redundancy. We think it might be possible that in later research, when our knowledge of algebraic hierarchical decompositions is more advanced than currently, we will return to this method or to some of its variants. However, for the time being we completely abandon $V \cup T$ decompositions for practical/computational applications.

Chapter 5

The Holonomy Decomposition

Now we turn to a different method with the following promising features: it does not just retain the information about the action on the state set (which is completely ignored in $V \cup T$ since it works with right regular representations), but the action is used in every aspect of the decomposition.

In order to state the theorem, which is somewhat different from the original Krohn-Rhodes Prime Decomposition Theorem (see Theorem 3.1), we first need to give a roadmap to the constructive proof.

5.1 Holonomy Decomposition Theorem

The holonomy decomposition originates from improving[1] Zeiger's method of proving the Krohn-Rhodes Theorem [Zei67, Gin68, Eil76, DN05]. This algorithm works by the detailed study of how the semigroup S of an automaton (A, X, δ) acts on certain subsets of A. It looks for groups induced by S^1 permuting some set of these subsets of A. These groups are called the *holonomy groups*. These groups are the building blocks for the components of the decomposition. As we go deeper in the hierarchy of the cascade composition we have components that act on a set of subsets each having smaller cardinality.

Sketch of the algorithm to obtain a holonomy decomposition: First calculate the set of images of transformations in S. From now on, let $\mathcal{I}$ denote this set extended by A itself and its singletons. On $\mathcal{I}$ there is a preorder relation called *subduction* defined. A subset P is subduction related to a subset Q if P is contained in the resulting set of acting by some $s \in S^1$ on Q, i.e. $P \subseteq Q \cdot s$. The mutual relation of elements induces an associated

[1] The improvement is that the components are decomposed in parallel whenever it is possible.

equivalence relation $P \equiv Q \iff P \leq Q$ and $Q \leq P$. The set of equivalence classes are partially ordered by the subduction relation. The set of equivalence classes and their partial order are called the *subduction picture*. The *tiles* $\mathcal{B}_P$ of a subset P ($P \in \mathcal{I}, |P| > 1$) are its maximal proper subsets in $\mathcal{I}$. The union of its tiles equals to P. The length of a longest strict path from a singleton to a subset P in the partial order of subduction equivalence classes defines the *height* of the subsets within the equivalence class of P. Consequently singletons have height 0. Equivalence classes with the same height are on the same hierarchical level. The height of an automaton $h(\mathcal{A})$ is the height $h(A)$ of its state set A, and this gives the number of hierarchical levels. The inclusion relation of the sets of tiles for each element $Q \in \mathcal{I}$ form the *tiling picture*. The holonomy group $\mathcal{H}_Q$ of Q is the group (arising from the action of the elements of S^1 on Q) permuting the tile set $\mathcal{B}_Q$ of Q. Then the holonomy decomposition component $(\mathcal{B}_i, \overline{\mathcal{H}_i})$ of one hierarchical level i is a permutation-reset ts and it is the direct product of the holonomy permutation groups $(\mathcal{B}_Q, \mathcal{H}_Q)$ belonging to the representative elements of equivalence classes with height i augmented with the constant mappings.

Theorem 5.1 (Holonomy Decomposition [Eil76, DN05]) *Let (A, S) be a finite transformation semigroup then (A, S) divides a wreath product of its holonomy permutation-reset transformation semigroups $(\mathcal{B}_1, \overline{\mathcal{H}_1}) \wr \cdots \wr (\mathcal{B}_h, \overline{\mathcal{H}_h})$, where h is the height of A.*

This strong formulation of the first part of the Krohn-Rhodes theorem is slightly different from the original since the components here are groups extended with constants and not simple groups and the divisors of the flip-flop. But these permutation-reset components can be easily decomposed into flip-flops and groups [KRT68]. Moreover the groups can be further decomposed into a series of simple groups using the Lagrange Coordinate Decomposition Theorem and Jordan-Hölder Theorem [Hal59, KRT68, DN05].

Note that the top level of the hierarchy for the holonomy decomposition is the component with the highest index, not 1. This is due to the importance of height function in determining the decomposition's structure.

Now the aim of the proof is clear and we can vaguely see the path leading to that goal, so it is time to dive into the details of the decomposition.

5.2 Relations of the Extended Set of Images

Here we consider relations defined on the image set of the characteristic semigroup. The structure determined by these relations form the skeleton for the decomposition.

5.2.1 The Extended Set of Images

For studying automata it is a common technique that we investigate how an automaton acts on the powerset 2^A of its state set A. Here we use a potentially smaller set of subsets $\mathcal{I} \subseteq 2^A$, the *extended set of images*. The extended set of images of A is defined by:

$$\mathcal{I} = \{A \cdot s \mid s \in S\} \cup \{A\} \cup \{\{a\} \mid a \in A\}$$

or more briefly

$$\mathcal{I} = \{A \cdot s \mid s \in S^I\} \cup \{\{a\} \mid a \in A\}$$

where I acts as the identity transformation on A.

In other words, $\mathcal{I}$ is basically the set of all distinct images of transformations in S^I and all the singletons of A.

Regarding the size of $\mathcal{I}$ the worst case is the full ts on A, when $\mathcal{I} = 2^A$, thus we can have at most 2^n elements.

5.2.2 Inclusion

As $\mathcal{I} \subseteq 2^A$ we naturally have the set-theoretical inclusion relation $(\mathcal{I}, \subseteq)$. Clearly, this relation is transitive, reflexive and antisymmetric, thus it is a partial order. Minimal elements are the singletons and the unique maximal element is A itself. The inclusion relation is independent of the action of the semigroup (or one can say, after seeing the subsequent relations, that the inclusion uses the identity transformation).

5.2.3 Image Relation

The fact that an element $P \in \mathcal{I}$ is an image of another element Q, is determined by a transformation of S. Therefore, the 'being an image of' relation can be formulated like:

$$P \trianglelefteq Q \iff \text{there exists } s \in S^I,\ P = Q \cdot s\ (P, Q \in \mathcal{I}) \tag{5.1}$$

This relation is transitive (combining transformations) and reflexive(identity transformation), i.e. preorder.

5.2.4 The Subduction Relation and the Skeleton

Combining the inclusion and the image relation we have a relation called the *subduction* relation given by

$$P \leq Q \iff \text{there exists } s \in S^I,\ P \subseteq Q \cdot s\ (P, Q \in \mathcal{I}), \tag{5.2}$$

i.e. we can transform Q to include P. Shortly written it is the relation combination:

$$(\mathcal{I}, \leq) = (\mathcal{I}, \subseteq) \circ (\mathcal{I}, \trianglelefteq).$$

The subduction relation is reflexive, since $P \subseteq P \cdot I$, and it is transitive, since if $P \subseteq Q \cdot s_1$ and $Q \subseteq R \cdot s_2$ then $P \subseteq R \cdot s_1 s_2$, thus $P \leq R$. Therefore subduction is a pre-order, and the pre-ordered $(\mathcal{I}, \leq)$ is called the *skeleton* of the ts (A, S).

As we use the monoid S^I the subduction relation can be considered as the generalization of the inclusion relation. If $P \subseteq Q$ then $P \subseteq Q \cdot 1$, therefore $P \leq Q$.

An element of S which shows the existence of the relation between two elements is called a *witness*. If P is subduction related to Q, then a witness for $P \leq Q$ is denoted by w_{PQ}, thus $P \subseteq Q \cdot w_{PQ}$.

5.2.5 Equivalence Classes

We also have an equivalence relation on $\mathcal{I}$ by taking the mutual subduction relation: $P \equiv Q \iff P \leq Q$ and $Q \leq P$. Equivalent elements of $\mathcal{I}$ have the same cardinality:

Lemma 5.2 *If $Q \equiv P,\ Q, P \in \mathcal{I}$ then $|Q| = |P|$.*

Proof: Suppose that $|Q| < |P|$ then $P \subseteq Q \cdot s$ is impossible for all $s \in S$ as there is no transformation of a finite set giving a bigger image set. □

Note that the converse is not generally true.

The set of equivalents of $Q \in \mathcal{I}$ is denoted by E_Q. If subduction relation is considered as a directed graph ($\mathcal{I}$ as the set of nodes, and there is an arrow from P to Q if $P \leq Q$) then the equivalence classes are exactly the strongly connected components.

Moreover, the (arbitrarily chosen) representatives of the equivalence classes $\mathcal{I}/\equiv$ (and thus the classes themselves) are partially ordered, since if $\overline{P}$ represents P and $P \leq Q$, then for appropriate $s, s', s'' \in S^1$, we have $P \subseteq Q \cdot s$, $\overline{P} \subseteq P \cdot s'$, and $Q \subseteq \overline{Q} \cdot s''$, implying $\overline{P} \subseteq \overline{Q} \cdot s'' s s'$, whence $\overline{P} \leq \overline{Q}$. By symmetry, it follows that $P \leq Q \iff \overline{P} \leq \overline{Q}$. The property of antisymmetry comes from the symmetry of the equivalence relation.

Also we write $P < Q$ if $P \leq Q$ but not $Q \leq P$. Thus, $P < Q \iff \overline{P} < \overline{Q}$.

5.2.6 Tiles and Tiling

We say P is a *tile* of Q, and write $P \prec Q$, if $P \subset Q$ and for all $Z \in \mathcal{I}$, $P \subseteq Z \subseteq Q$ implies $Z = P$ or $Z = Q$. It follows that $P < Q$ as P is a proper subset of Q.

The set of tiles of Q for any $|Q| > 1$ is denoted by $\mathcal{B}_Q = \{P \in \mathcal{I} \mid P \prec Q\}$. Since $\mathcal{I}$ contains the singletons and singletons contained in Q are subduction related to Q at least by the identity transformation, therefore for $|Q| > 1$, Q equals the union of its tiles, i.e. $Q = \bigcup_{P \in \mathcal{B}_Q} P$. For this reason the covering $\mathcal{B}_Q$ is called the *tiling* of Q. Note that the tiles may overlap.

5.2.7 Tile Chains

A *tile chain* is a sequence of elements of $\mathcal{I}$, where successive elements are in tile of relation: $\{a\} = B_1 \prec \ldots \prec B_k = A$, $k \leq h$. As we will see later tile chains starting from the singleton $\{a\}$ can be considered as lifts for state a.

5.2.8 Height of a Subset

The *height* of a member Q of $\mathcal{I}$ in the skeleton $\mathcal{I}, \leq$ is given by the function $h : \mathcal{I} \to \mathbb{Z}$, which is defined by $h(Q) = 0$ if Q is a singleton, and for $|Q| > 1$, $h(Q)$ is defined by the length of the longest chain(s) in the skeleton starting from a non-singleton set and ending in Q:

$$h(Q) = \max_i (Q_1 < \cdots < Q_i = Q),$$

where $|Q_1| > 1$. The height of (A, S) is $h = h(A)$.

Lemma 5.3 *$P \equiv Q \Rightarrow h(P) = h(Q)$.*

Proof: Suppose that $h(P) = i$, $h(Q) = j$ and $i < j$. Therefore we have chains like $P_1 < \cdots < P_i = P$ and $Q_1 < \cdots < Q_j = Q$. But following from the equivalence we have the $Q_1 < \cdots < Q_{j-1} < P$, contradicting that $h(P) < j$. □

Lemma 5.4 *If $h(P) = h(Q)$ and $\exists s \in S^I$ such that $P \cdot s = Q$, then $P \equiv Q$.*

Proof: The proof is indirect: Suppose $Q < P$, then we can append P to a strict maximal subduction chain of Q (where the height Q comes from), thus getting $h(P) > h(Q)$ contradicting our original assumption. □

5.3 Components

5.3.1 Holonomy Groups

Define H_Q to be the set of permutations of $\mathcal{B}_Q$ induced by elements of $s \in S^I$. That is, if for $s \in S^I$, the function $s_Q : \mathcal{I} \to \mathcal{I}$ defined by $s_Q(z) = z \cdot s = \{a \cdot s \mid a \in z\}$ $(z \in \mathcal{I})$ restricts to $s_Q : \mathcal{B}_Q \to \mathcal{B}_Q$ and permutes the elements of $\mathcal{B}_Q$, then $s_Q \in H_Q$. H_Q is called the *holonomy group* of Q in (A, S), and clearly H_Q divides S, and $(\mathcal{B}_Q, H_Q)$ is a permutation group and it is called the *holonomy permutation group* of Q.

5.3.2 Holonomy Permutation-Reset Transformation Semigroups

Although the holonomy groups are building blocks for the semigroup being decomposed, but they are not sufficient for the construction, since we need to represent the possible collapsing of states, not just permutations. Therefore we extend a holonomy group $(\mathcal{B}_Q, H_Q)$ with constant mappings $C_P,\ P \in \mathcal{B}_Q$. Thus, if $(\mathcal{B}_Q, H_Q)$ is a holonomy group, then $(\mathcal{B}_Q, \overline{H_Q})$ is a *holonomy transformation semigroup.*

The height values define hierarchical levels. Since there can be more than one equivalence class on the same level, components are composite. For each i $(1 \leq i \leq h)$, define $(\mathcal{B}_i, \mathcal{H}_i)$ to be the direct product of the holonomy permutation groups of the height i representatives in $\mathcal{I}$. Then $\mathcal{B}_i = \prod_{h(\overline{P})=i} \mathcal{B}_{\overline{P}}$ and $\mathcal{H}_i = \prod_{h(\overline{P})=i} H_{\overline{P}}$. Then $(\mathcal{B}_i, \mathcal{H}_i)$ is a permutation group and $(\mathcal{B}_i, \overline{\mathcal{H}_i})$ is the associated holonomy permutation-reset transformation semigroup obtained by adjoining all constant maps taking values in $\mathcal{B}_i$. We denote elements of $\mathcal{B}_i$ by boldface variables $\mathbf{B}_i$. We also talk about *positions* in $\mathcal{B}_i$ according to the components (equivalence classes) of the direct product. Using projection maps $\pi_{\overline{P}}$ indexed by the class representatives $\pi_{\overline{P}}(\mathbf{B}_i)$ denotes the element of $\mathbf{B}_i$ in the $\overline{P}$-position, where $P \in \mathcal{I},\ h(P) = i,\ \mathbf{B}_i \in \mathcal{B}_i$. Although any element identifies its equivalence class, we use the representative for that purpose.

5.4 Mappings on $\mathcal{I}$

5.4.1 Isomorphisms of Holonomy Groups within a Subduction Equivalence Class

First we have to show that the choice of the representative is really arbitrary, i.e. the holonomy groups of elements of a class are isomorphic. Several constructs defined here are used later.

Lemma 5.5 *If $Q \equiv P$ $(|Q| > 1)$, and w_{PQ} (resp. w_{QP}) is a witness for $P \leq Q$, then w_{PQ} is a bijective mapping from Q to P (resp. P to Q).*

Proof: By Lemma 5.2, $Q \equiv P$ implies that $|Q| = |P|$. Thus by finiteness $Q \subseteq P \cdot w_{QP}$ implies $Q = P \cdot w_{QP}$. □

Lemma 5.6 *If $Q \equiv P$ $(|Q| > 1)$, w_{PQ} and w_{QP} are witnesses respectively, then $w_{QP}w_{PQ}$ permutes the elements of P (and $w_{PQ}w_{QP}$ permutes the elements of Q).*

Proof: According to the definition of $\equiv$, $P \leq Q$ so $P \subseteq Q \cdot w_{PQ}$. Substituting $P \cdot w_{QP}$ (as $Q \leq P$, $Q \subseteq P \cdot w_{QP}$) for Q gives $P \subseteq P \cdot w_{QP} \cdot w_{PQ}$. Since P is finite, $P \subseteq P \cdot w_{QP}w_{PQ}$ implies $P = P \cdot w_{QP}w_{PQ}$ (no transformations

yield bigger images), i.e. $w_{QP}w_{PQ}$ permutes the elements of P. (The proof of the other direction is similar.) $\square$

Since the subduction relation is a generalization of the set theoretic inclusion relation, if P is not related to Q then it follows that P is not a subset of Q (but not in the opposite way). This observation is used in the proof of the following lemma.

Lemma 5.7 *If s is a bijective mapping $P \mapsto Q$ then s is also a bijective mapping $\mathcal{B}_P \mapsto \mathcal{B}_Q$.*

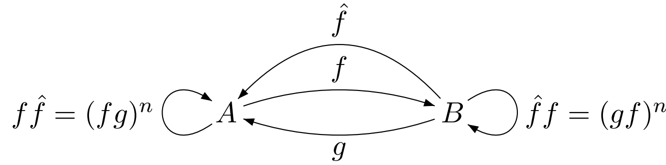
Proof: Let $Z \in \mathcal{B}_P$ and $Z' = Z \cdot s$. Suppose that Z' is not a tile of Q, i.e. $\exists Z'' \in \mathcal{B}_Q$ such that $Z' \subseteq Z'' \cdot u$ for some $u \in S^I$. Then using the inverse mapping of s we get $Z^* = Z'' \cdot \hat{s}$. But the fact that $Z \subseteq Z^* \cdot su\hat{s}$ contradicts the original assumption that Z is a tile. $\square$

Remark 5.8 *Since bijections have inverses, bijective mappings between tilings map tiles to equivalent tiles.*

We construct bijective mappings between equivalent elements that show isomorphism of their holonomy permutation groups. In order to do that we need a general lemma on bijections between finite sets (Fig. 5.9).

Lemma 5.9 *Let $f : A \to B$, $g : B \to A$ bijective mappings on finite sets A, B. Take $n > 1$ such $(fg)^n = 1_A$, the identity permutation of A, then $(gf)^n = 1_B$.*

Proof: The inverse of f is $\hat{f} = g(fg)^{n-1}$, thus $f\hat{f} = 1_A$. Take arbitrary elements $a \in A$, $b \in B$, such that f maps a to b and $\hat{f}$ maps b to a. Now consider $\hat{f}f = g(fg)^{n-1}f = (gf)^n$, it maps b to b, so $\hat{f}f = 1_B$. $\square$



The point of the lemma is the synchronicity of the two directions, identity permutation appears for the same n.

Lemma 5.10 *If $Q \equiv P$ then $(\mathcal{B}_Q, H_Q)$ is isomorphic to $(\mathcal{B}_P, H_P)$.*

Proof: To prove the isomorphism we have to find a bijective homomorphism. Let w_{QP}, w_{PQ} be witnesses for the equivalence, then they are bijections, thus $w_{PQ}w_{QP}$ permutes the elements of Q, and similarly $w_{QP}w_{PQ}$ permutes those of P. By Lemma 5.7, it follows that they permute the corresponding tile sets as well. Take $n > 1$ such $(w_{PQ}w_{QP})^n$ is the identity permutation of $\mathcal{B}_Q$.

Let $\tau = w_{QP}(w_{PQ}w_{QP})^{n-1}$, so τ is the inverse of w_{PQ}. Then, according to Lemma 5.9, $w_{PQ}\tau$ acts as the identity on $\mathcal{B}_Q$, and also τw_{PQ} acts as the identity on $\mathcal{B}_P$.

Tiles For a tile $Z \in \mathcal{B}_Q$, $Z \mapsto Z \cdot w_{PQ} \in \mathcal{B}_P$ is bijective onto $\mathcal{B}_P$ with inverse $Z' \mapsto Z' \cdot \tau$,

Permutations For a permutation $s_Q \in H_Q$, $s_Q \mapsto \tau s_Q w_{PQ}$ is bijective onto H_P with inverse $s_P \mapsto w_{PQ} s_P \tau$,

Actions For a permutation $s_Q \in H_Q$ and a tile $Z \in \mathcal{B}_Q$, for the map $s_Q \mapsto \tau s_Q w_{PQ} \in H_P$ we get $(Z \cdot w_{PQ}) \cdot \tau s_Q w_{PQ} = (Z \cdot s_Q) \cdot w_{PQ}$,

Products For two permutations $s_Q, t_Q \in H_Q$ using the same map as before we get $\tau(s_Q t_Q)w_{PQ} = \tau s_Q w_{PQ} \tau t_Q w_{PQ}$ as $w_{PQ}\tau$ is the identity on $\mathcal{B}_Q$.

Hence, we have an isomorphism of permutation groups. □

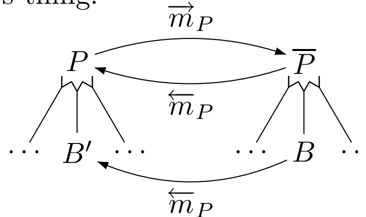
5.4.2 Moving within an equivalence class

With the help of bijections used in the proof of Lemma 5.10 we can define isomorphism mappings from the holonomy permutation group of $P \in \mathcal{I}$ to that of the corresponding equivalence class representative $\overline{P}$ and back. These are used in lifting states and transformations when establishing the homomorphism. Let $\overline{P}$ is an arbitrary chosen representative of the equivalence class of $P \in \mathcal{I}$. Define $\overrightarrow{m}_P = w_{\overline{P}P}$, thus $\overline{P} = P \cdot \overrightarrow{m}_P$, i.e. mapping from $(\mathcal{B}_P, H_P)$ *to* the holonomy group $(\mathcal{B}_{\overline{P}}, H_{\overline{P}})$ of its representative $\overline{P}$. For the other direction, the mapping *away* from the representative $\overleftarrow{m}_P = w_{P\overline{P}}(w_{\overline{P}P}w_{P\overline{P}})^{n-1}$. It is immediate that $\overleftarrow{m}_{\overline{P}} = \overrightarrow{m}_{\overline{P}} = 1_{\overline{P}}$.

By shifting our attention from the sets to their tilings we define the following "selector function":

$$\sigma(P, B) = B \cdot \overleftarrow{m}_P \text{ where } P \in \mathcal{I},\ B \in \mathcal{B}_{\overline{P}}$$

which selects a tile from the tiling of P based on a tile of the equivalence class representative's tiling.



We also define the inverse selector function by:

$$B = \hat{\sigma}(P, B') = B' \cdot \overrightarrow{m}_P \text{ where } P \in \mathcal{I},\ B' \in \mathcal{B}_P$$

which chooses a tile of $\overline{P}$ based on a tile of P.

5.5 Lifting the State Set

The state set of the cascaded product is clearly bigger than the original automaton's, so one might think that there are cascaded states which have no counterparts in the original state set. We show that this is not true, as every cascaded state can be mapped down to an original state, and this mapping is onto. This small result is new, and it simplifies the proof, since we don not need to handle the exceptional case, when there is no preimage. We also give a mapping which gives at least one cascaded state as a lift for an original state.

5.5.1 Successive Approximation of States

Due to the hierarchical nature of the wreath product we can approximate the original automaton's behavior by considering only some hierarchical levels starting from the top level. Going top-down means more detailed approximation. This involves the approximation of states by a series of subsets of the state set, and ultimately the mapping $\eta : \mathcal{B}_1 \times \ldots \times \mathcal{B}_h \to A$.

We define $\eta_i : \mathcal{B}_i \times \ldots \times \mathcal{B}_h \to \mathcal{I}$ inductively as i goes from h to 1 by

$$\eta_h(\mathbf{B}_h) = \sigma(A, \mathbf{B}_h) = \mathbf{B}_h$$

which is a tile of X since the top level is not composite, and letting $P = \eta_{i+1}(\mathbf{B}_{i+1}, \ldots, \mathbf{B}_h)$ which we suppose already well-defined for $h \geq i+1 > 1$, we define

$$\eta_i(\mathbf{B}_i, \ldots, \mathbf{B}_h) = \begin{cases} P & \text{if } h(P) < i \\ \sigma(P, B) & \text{if } h(P) = i \text{ and } B = \pi_{\overline{P}}(\mathbf{B}_i). \end{cases}$$

In the first case we "jump over" the ith level as the approximation is already gone forward by choosing a tile with small cardinality at some upper level. Therefore $\mathbf{B}_i$ can have no affect on the value of η_i. In the second case we are on the right hierarchical level, thus we can apply the selector function. Observe that the only one element of $\mathbf{B}_i$ acts in the selection, and this is true more generally: on all levels at most one position of $\mathbf{B}_i$ can affect the value of η_i.

The selector function gives a tile of P, therefore either $\eta_i(\mathbf{B}_i, \ldots, \mathbf{B}_h) \prec \eta_{i+1}(\mathbf{B}_{i+1}, \ldots, \mathbf{B}_h)$ or they are equal. Therefore by omitting equal elements we get a chain of tiles:

$$\{a\} = B_1 \prec \ldots \prec B_k = A, \; k \leq h.$$

Since in all cases $h(\eta_i(\mathbf{B}_i, \ldots, \mathbf{B}_h)) < i$, η_1 gives a singleton. By the unique element of this singleton we define the value of $\eta : \mathcal{B}_1 \times \ldots \times \mathcal{B}_h \to A$.

5.5.2 Lifting the States

"The road up and down is one and the same."
Heraclitus, DK22B60

We have seen that every element of $\mathcal{B}_1 \times \cdots \times \mathcal{B}_h$ can be mapped down to a singleton by η. We also have to show that η is surjective, which in this context means that every element $a \in A$ has at least one lift in $\mathcal{B}_1 \times \cdots \times \mathcal{B}_h$. To accomplish that, we choose an arbitrary state $a \in A$ and by calculating η bottom-up instead of top-down (much like as an inverse) we construct a $(\mathbf{B}_1, \ldots, \mathbf{B}_h)$ such that $\eta(\mathbf{B}_1, \ldots, \mathbf{B}_h) = a$.

As shown before, the successive approximation has the very nice property, that at each level only one position may affect the final result. Therefore we need the following notation for focusing on one class and discarding the others with the same height. Suppose that $B \prec P$, $P = \overline{P}$ for a $P \in \mathcal{I}$, $h(P) = i$. Then we denote by $[B]_P$ any arbitrary element of $\mathcal{B}_i$ containing tile B in the P-position. Similarly for transformations, if $g \in H_P$ then we write $[g]_P$ for any arbitrary element of $\mathcal{H}_i$ containing g in the $\overline{P}$-position, and identity elements in all other positions.

We create a chain of tiles: $\{a\} = B_1 \prec \ldots \prec B_k = A$, $k \leq h$ (like the stages of the successive approximation). Then we map these tiles to representative tilings, thus they will be selected during the successive approximation:

$$\mathbf{B}_i = [\hat{\sigma}(B_{j+1}, B_j)]_{\overline{B_{j+1}}} \text{ where } h(B_{j+1}) = i.$$

We also have to fill the levels which are jumped over. For such a level i any fixed but arbitrary $\mathbf{B}_* \in \mathcal{B}_i$ is suitable.

5.6 Lifting the Semigroup

5.6.1 Lifting Transformations

For the constructive proof the explicit description of the dependency functions are not needed, since it is enough to consider only the action on only one particular state lift. Recall that an element of the wreath product is given by describing its component actions. Thus to specify lift $\tilde{s}$ of an $s \in S$ to the wreath product we need to give appropriate functions $\tilde{s}_i : \mathcal{B}_{i+1} \times \cdots \times \mathcal{B}_h \to \overline{\mathcal{H}_i}$ for $i = h, \ldots, 1$. (For $i = h$, $\tilde{s}_h$ is just an element of $\overline{\mathcal{H}_h}$.) Such an h-tuple $(\tilde{s}_1, \ldots, \tilde{s}_h)$ of functions determines a transformation in the wreath product. It is hard to give a nice closed formula for those functions, instead we describe an algorithm that gives the transformations for any particular $(\mathbf{B}_1, \ldots \mathbf{B}_h)$.

For defining a lift for a member s of S to the wreath product we use a simple trick. We do the successive approximation $(\mathbf{B}_1, \ldots, \mathbf{B}_h)$ and at each

stage of we apply s to P getting a set $P \cdot s \in \mathcal{I}$. Then we choose transformations on the levels above to approximate $P \cdot s$. Roughly, if collapsing occurs in the action of s then we use a constant map to a tile approximating the resulting set. When determining these constant maps we may have freedom to choose from more than one tiles if more than one contains $P \cdot s$. Then the choice B is arbitrary but fixed. If s acts as a permutation on P, then we choose a permutation from the corresponding holonomy group. At all level the following is true:

$$\eta_i(\mathbf{B}_i, \ldots, \mathbf{B}_h) \cdot s \subseteq \eta_i(\mathbf{B}_i \cdot \tilde{s_i}, \ldots, \mathbf{B}_h \cdot \tilde{s_h})$$

yielding the final equality that $\eta(\mathbf{B}_1, \ldots, \mathbf{B}_h) \cdot s = \eta((\mathbf{B}_1, \ldots, \mathbf{B}_h) \cdot \tilde{s})$, since the approximation gives singletons in the end.

The construction of the lift goes inductively. On the top level we define

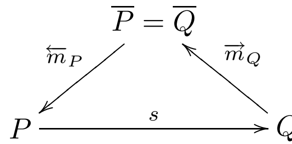
$$\tilde{s}_h = \begin{cases} \text{constant } B & \text{if } A \cdot s \subset A, \text{ and } A \cdot s \subseteq B \prec A \\ s_A & \text{if } A \cdot s = A. \end{cases}$$

Going down let's $P = \eta_{i+1}(\mathbf{B}_{i+1}, \ldots, \mathbf{B}_h)$ and $Q = \eta_{i+1}(\mathbf{B}_{i+1} \cdot \tilde{s_{i+1}}, \ldots, \mathbf{B}_h \cdot \tilde{s_h})$, then

$$\tilde{s}_i = \begin{cases} \text{constant } [\hat{\sigma}(Q, B)]_{\overline{Q}} & \text{if } P \cdot s \subset Q, h(Q) = i, \text{ and } P \cdot s \subseteq B \prec Q, \\ [\overleftarrow{m}_P s \overrightarrow{m}_Q]_{\overline{Q}} & \text{if } P \cdot s = Q \text{ and } h(P) = h(Q) = i, \\ \text{arbitrary } t \in \mathcal{H}_{\overline{Q}} & \text{if } h(Q) < i. \end{cases}$$

In the first case, collapsing of states happens, since $P \cdot s \subset Q$, thus we choose a tile B of Q which contains $P \cdot s$, and by using the inverse selector function we pick up a constant map resetting to a tile of $\overline{Q}$ to make sure that B will be given by η_i. Clearly this constant map lies in $\overline{\mathcal{H}_i}$.

In the second case, we have $h(P) = i = h(Q) = h(P \cdot s) \geq 1$, whence $Q = P \cdot s \equiv P$. Therefore $\overline{P} = \overline{Q}$. This implies that $\overleftarrow{m}_P s \overrightarrow{m}_Q$ represents an element of $H_{\overline{P}}$, so that $[\overleftarrow{m}_P s \overrightarrow{m}_Q]_{\overline{P}} \in \mathcal{H}_i$.



The last case applies on levels which are jumped over.

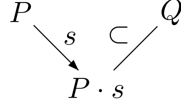
In all cases, $\tilde{s}_i(\mathbf{B}_{i+1}, \ldots, \mathbf{B}_h) \in \overline{\mathcal{H}_i}$ as required.

5.6.2 Verifying the division

We have to show that for any stage of the successive approximation, where P approximates the state and Q the transformed state, $P \cdot s \subseteq Q$ holds. This clearly holds for the top level, since $\mathbf{B}_h \cdot s \subseteq \mathbf{B}_h \cdot \tilde{s_h}$. Now assuming inductively that it holds for P and Q, we establish it for the next stage. We shall consider three cases:

Case 1 : $P \cdot s \subset Q$, $h(P) \leq i$ and $h(Q) = i$.

If $h(P) < i$ then $P' = \eta_i(\mathbf{B}_i, \ldots, \mathbf{B}_h) = P$ by definition of η_i since $h(P) < i$. While if $h(P) = i$ then $P' \prec P$.



Now $Q' = \eta_i(\mathbf{B}_i \cdot \tilde{s}_i, \ldots, \mathbf{B}_h \cdot \tilde{s}_h) = \sigma(Q, \hat{\sigma}(Q, \pi_{\overline{Q}}(\mathbf{B}_i)))$
$= (\mathbf{B}_i \cdot \tilde{s}_i)_{\overline{Q}} \cdot \overleftarrow{m}_Q$ since $h(Q) = i$,
$= (B \cdot \overrightarrow{m}_Q)\overleftarrow{m}_Q$, where $P \cdot s \subseteq B \prec Q$ according to the definition of $\tilde{s}_i$
$= B$.
Therefore $P' \cdot s \subseteq P \cdot s \subseteq B = Q'$, no matter the height of P.

Case 2 : $h(P) = i$, $h(Q) = i$, $P \cdot s = Q$. We have $P \equiv Q$, so $\overline{P} = \overline{Q}$. This implies that s maps $\mathcal{B}_P$ bijectively onto $\mathcal{B}_Q$.

$$P \xrightarrow{s} Q$$

Again $P' = \eta_i(\mathbf{B}_i, \ldots, \mathbf{B}_h) = (\mathbf{B}_i)_{\overline{P}} \cdot \overleftarrow{m}_P \prec P$ and
$Q' = \eta_i(\mathbf{B}_i \cdot \tilde{s}_i, \ldots, \mathbf{B}_h \cdot \tilde{s}_h)$
$= (\mathbf{B}_i \cdot \tilde{s}_i)_{\overline{Q}} \cdot \overleftarrow{m}_Q$
$= (\mathbf{B}_i)_{\overline{P}} \cdot (\overleftarrow{m}_P s \overrightarrow{m}_Q)_{\overline{P}} \overleftarrow{m}_Q$ since $\overline{P} = \overline{Q}$ and by definition of $\tilde{s}_i$.
$= (\mathbf{B}_i)_{\overline{P}} \cdot \overleftarrow{m}_P s \overrightarrow{m}_Q \overleftarrow{m}_Q$
$= (\mathbf{B}_i)_{\overline{P}} \cdot \overleftarrow{m}_P s (\overrightarrow{m}_Q \overleftarrow{m}_Q)$
$= (\mathbf{B}_i)_{\overline{P}} \cdot \overleftarrow{m}_P s$ since $\overrightarrow{m}_Q \overleftarrow{m}_Q$ acts as the identity on B_Q.
Therefore $P' \cdot s = (\mathbf{B}_i)_{\overline{P}} \cdot \overleftarrow{m}_P \cdot s = Q'$.

Case 3 : $h(P) < i$ and $h(Q) < i$. Then by definition of η_i, $P' = P$ and $Q' = Q$, so the conclusion holds by induction hypothesis.

By induction we conclude that $\eta_i(\mathbf{B}_i, \ldots, \mathbf{B}_h) \cdot s \subseteq \eta_i(\mathbf{B}_i \cdot \tilde{s}_i, \ldots, \mathbf{B}_h \cdot \tilde{s}_h)$ for all i $(1 \leq i \leq h)$, all $s \in S$ and all $(\mathbf{B}_1, \ldots, \mathbf{B}_h)$.

Moreover, lifts of distinct members of A are distinct since η is a function; and lifts of distinct members of S are distinct: If $s_1 \neq s_2$ $(s_1, s_2 \in S)$ then there is an $a \in A$ such that $a \cdot s_1 \neq a \cdot s_2$. Taking a lift $\tilde{a}$ of a we have $\eta(\tilde{a} \cdot \widetilde{s_i}) = \eta(\tilde{a}) \cdot s_i = a \cdot s_i$, but these are distinct for $i = 1, 2$, therefore the lifts $\widetilde{s_1}$ and $\widetilde{s_2}$ are also distinct. This establishes the division and proves the Holonomy Theorem. □

5.6.3 Dependency Functions

A transformation in the wreath product of the holonomy components is a tuple of functions: $\tilde{s} = (f_1, \ldots, f_h)$ (recall Section 2.3). These functions describe how the component action on one particular level is determined by the states of the levels above; briefly they delineate the hierarchical dependencies between the levels.

For the lifts of the generators the dependency functions are now fully defined in all cases, since we know their values. Based on the lifting method we have the following considerations.

Identity as independence. If the value of a dependence function is the identity on a certain level, then the component's state remains the same, therefore the can be considered independent from the other coordinates above in respect of the transformation defined by the dependence functions[2].

Levels jumped over are independent. For the levels jumped over by the successive approximation we can define the value as the identity. In fact any other action can be chosen, but this is consistent with the idea of independence.

Composite components have independent parts. In the case of a composite component only one component takes actual role in the transformation.

5.6.4 The Circuitry of the Wreath Product

It is a well-known psychological fact in mathematics that it is a lot easier to understand isomorphisms rather than automorphisms. It is more natural to relate two separate things, which happen to be the same except their description, than relating something to itself in a peculiar way. Since in the second case we have to keep track on which side of the morphisms we are actually. Quite similar thing happens when we want to use cascaded machines in order to understand the original automaton's behavior. We have to separate cascaded machine with its circuitry, describe it independently of the original automaton. Only after that we can characterize the morphism between them by giving the the mappings of the coordinates onto the original automaton.

If the transformations of the wreath product, the tuples of fully defined dependency functions are available, then we have everything to get the cascaded product to work. Equations 2.1 and 2.2 show how elements can be multiplied by using function composition.

[2]One might say that it is just a special kind of dependence, and that is right. The reason why we call it independence comes from implementational issues, since if it maps to the identity, then it does not need to be stored.

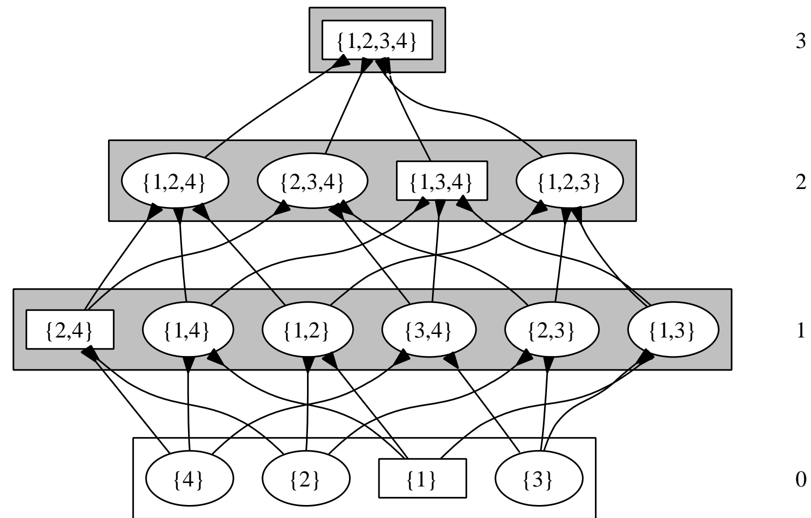


Figure 5.1: The tiling picture of $\mathcal{T}_4$, the full transformation semigroup on 4 points. The big boxes denote equivalence classes and within them sets with boxes denote the arbitrary choosen equivalence class representatives. The gray shade indicates the existence of nontrivial holonomy groups. The arrows denote the tile of relation.

The action on the state lifts is also simple: we need to apply the dependency functions, thus we get a tuple of component actions. The we apply the actions in the component which yields a new state lift.

5.7 Examples

5.7.1 The Tricks of Tiling

Tiling may look like a simple and intuitive concept, but this can be – in general – a bit misleading. Therefore we present some examples to show some subtle issues that are crucial properties for computational implementation.

Tiling in the Full Transformation Semigroup

Again, we start with the full transformation semigroup due to its special role (namely to be the 'biggest') among finite transformation semigroups. We will see that its tiling picture is quite regular, and this regularity can be deceiving with careless generalization. By regular here we mean that for the tiling picture of the full ts the following statements are true:

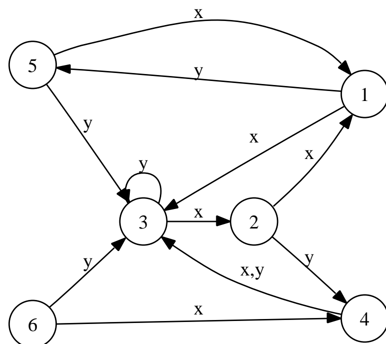


Figure 5.2: An example automaton from Eilenberg's book [Eil76] Exercise 9.6. The generators $\{x = (3\ 1\ 2\ 3\ 1\ 4), y = (5\ 4\ 3\ 3\ 3\ 3)\}$.

- $|B_Q| = |Q|$, i.e. the number of tiles equals to the cardinality of the tiled set.
- $|Q| = h(Q) + 1$, thus the height numbers correspond to cardinalities.
- $h(P) = h(Q) - 1$ for all $P \in B_Q$, i.e. there are no cross level tiles.
- $P \prec Q \implies P \trianglelefteq Q$, i.e. all tiles are images of the tiled sets.

where $Q \in \mathcal{I}$, $|Q| \geq 2$, i.e. Q is a nonsingleton element of $\mathcal{I}$. These properties can be checked on Fig. 5.1. Since these properties might lure us to make assumptions about easy solutions in a computational implementation, in the following we show examples breaching these conditions.

For the full ts the holonomy decomposition gives the compact wreath product (see Section 4.2.1) calculated by Eilenberg [Eil76].

5.7.2 Cross Level Tiles

We talk about cross level tiles when the difference in the height values is bigger than one between the tile and the tiled set. Looking at the example on Fig. 5.2 and 5.3 shows that there are indeed such tiles. $\{6\} \prec \{1, 2, 3, 4, 5, 6\}$ is somewhat 'artificial', since $\{6\}$ is not an image of the characteristic semigroups' elements. this shows the necessity of the singletons in $\mathcal{I}$. Otherwise in general, we will not be able to tile the subsets in the set of images. But not only singletons can be cross level tiles, $\{3, 4, 5\} \prec \{1, 2, 3, 4, 5, 6\}$ is another example, and in this case tile is a real image. The cascaded product is built from the following components:

$$(\mathbf{3}, \overline{S_3}) \wr (\mathbf{2}, \overline{1_2}) \wr (\mathbf{3}, \overline{1_3}).$$

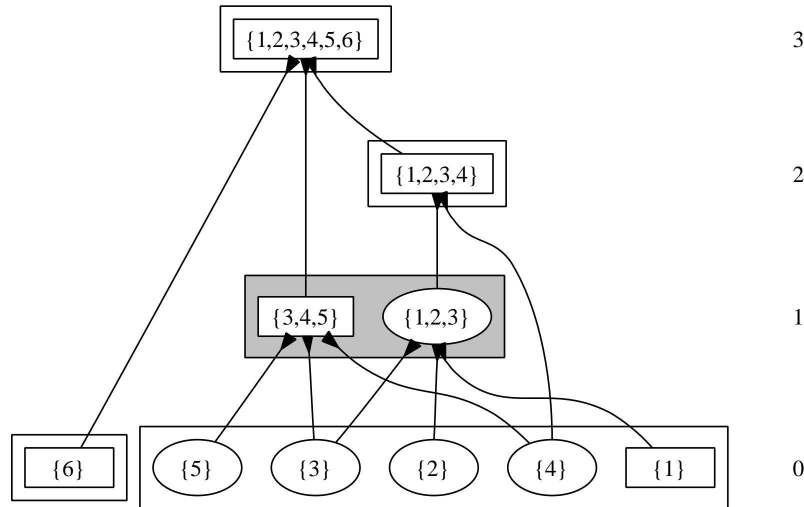


Figure 5.3: The tiling picture of Eilenberg's Exercise 9.6. automaton.

5.7.3 Nonimage Tiles

Continuing the previous topic, now we are interested in tiles, that are not images of the tiled set. This might be a crucial issue in the efficient calculation of the holonomy decomposition. The problem looks simple, since we just apply the generators to the set to be tiled, and record the maximal images. Unforutnately this does not work due to the fact that there can be a tiling with a set of tiles with the property, that none of them is an image of the tiled set, even if the tiled set has a nontrivial holonomy group. A carefully crafted example revelas this fact. We need the following generators:

$x = (1\ 2\ 3\ 1\ 1\ 1)$ creates the image $\{1,2,3\}$.

$\{y = (4\ 4\ 4\ 5\ 4\ 6), z = (4\ 4\ 4\ 5\ 6\ 4)\}$ give the image $\{4,5,6\}$ and for the generator set (a transposition and a cycle) for the holonomy component S_3 on the image.

$u = (4\ 4\ 4\ 4\ 5\ 5)$ This and the nontrivial holonomy group generate the images with cardinality 2.

$v = (4\ 4\ 4\ 1\ 2\ 3)$ This maps $\{4,5,6\}$ to $\{1,2,3\}$.

$w = (2\ 3\ 1\ 4\ 4\ 4)$ Just to make $H_{\{1,2,3\}}$ be nontrivial.

The decomposition is the following:

$$(\mathbf{2}, \overline{1_2} \wr \overline{(\mathbf{2}, C_2) \times \mathbf{3}, C_3)} \wr \overline{(\mathbf{2}, C_2) \times (\mathbf{3}, C_3)} \wr (\mathbf{4}, \overline{S_3}) \wr (\mathbf{2}, \overline{1_2})$$

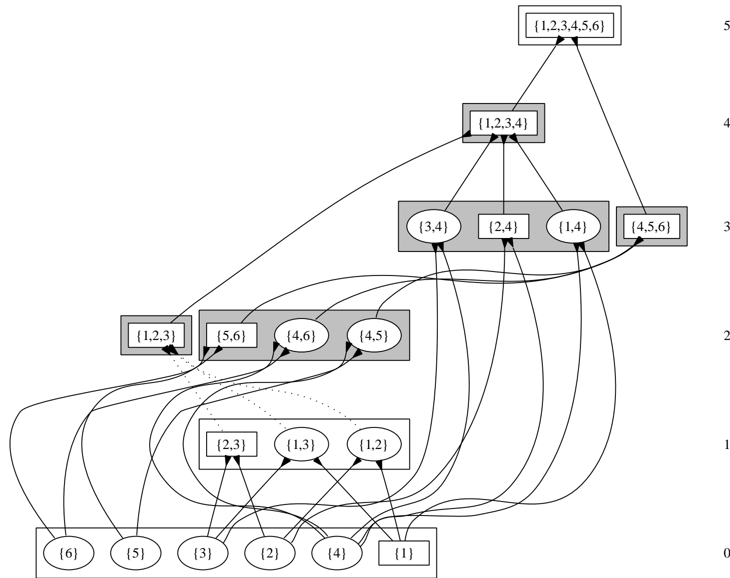


Figure 5.4: A tiling picture of an automaton, which shows that it is possible that a tiled set can have only nonimage tiles. For the description of the generators see the text.

It is worth noting that the fourth level component has the symmetric group on 3 points as its holonomy group, but has 4 tiles. The explanation is that it acts as the identity on the fourth state.

5.7.4 Strict Subduction for Sets with the Same Cardinality

Contrary to what the example of full ts suggests, it is possible to have strict subduction relation between sets with the same cardinalities. We need only a transformation which has no inverse transformation in the semigroup regarding that subset. Let's consider a very simple example.

$x = (1\ 2\ 1\ 2)$ maps $\{3,4\}$ to $\{1,2\}$ (and creates the image $\{1,2\}$).

$y = (3\ 3\ 3\ 4)$ maps $\{1,2\}$ only to $\{3\}$ (but also creates the image $\{3,4\}$).

The partial order set of the equivalence classes can be seen on Fig. 5.5.

5.7.5 Tile Chains

We hinted that the statelifts are basically tile chains starting from the corresponding singleton of the original state, though they are encoded into a

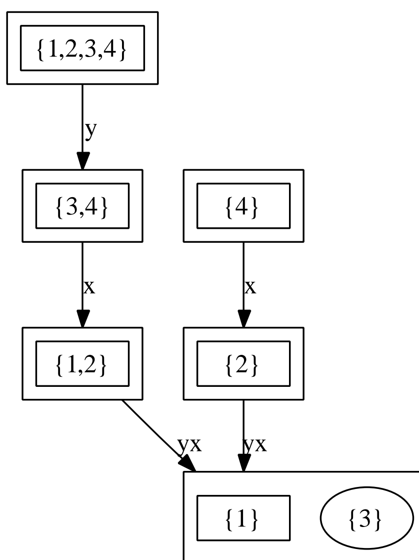


Figure 5.5: The image relation partial order of the equivalence classes for the ts generated by $\{x = (1\ 2\ 1\ 2), y = (3\ 3\ 3\ 4)\}$. The labels of the arrows denote witnesses for the image relation.

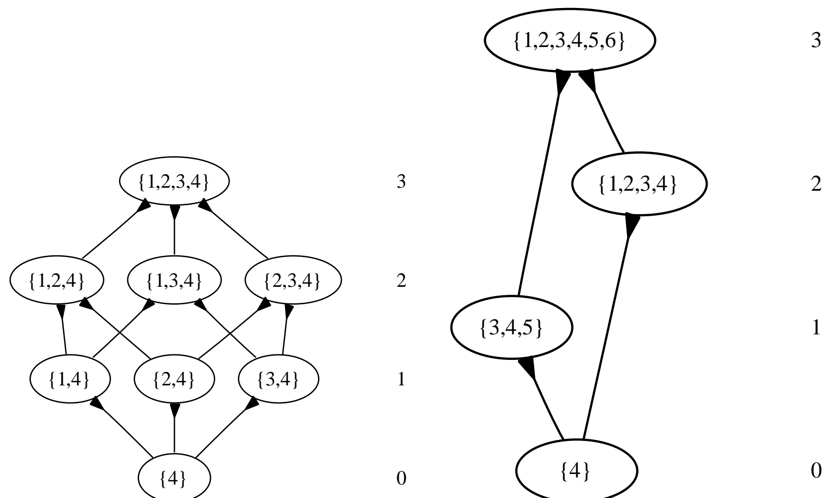


Figure 5.6: The tile chains starting from the singleton $\{4\}$ and in the skeleton of $\mathcal{T}_4$ (on the left) and Eilenberg's Exercise 9.6. (on the right). In the case of $\mathcal{T}_4$ the state 4 has 6 lifts (6 different paths can be chosen to reach the full state set), in the other case there are only 2 state lifts.

subduction chain in respect to the arbitrary chosen equivalence representatives. Therefore, the number of statelifts equals to the number of tile chains starting from the singleton set of the state. See Fig. 5.6

5.8 Summary

We described the holonomy method as a constructive proof, focusing on details that are crucial for a computational implementation. We also made efforts to improve the notation, though this might be difficult to judge objectively. It is important to notice that there is a slight change in the way of thinking, departing from the mathematical viewpoint which is mainly interested in what is proven, proceeding to the more practical view, which says that the cascade composition is useful device and it is worth studying in itself.

Chapter 6

Implementational Details of the Holonomy Decomposition

Though we have a detailed constructive proof of the holonomy decomposition, it is still far from a working computational implementation. Constructive proofs usually provide a clear constructive description of the main steps of the algorithm to obtain a decomposition but say nothing about how the steps should be carried out and how the mathematical objects involved should be represented computationally. Moreover, they never consider the computational feasibility: the space and time complexity of the required calculations. The main concern of an efficient implementation is to try to avoid combinatorial explosions. Therefore the problematic points of the algorithms are where the $\forall, \exists$ symbols appear in proofs.

6.1 Related Software Packages

There are many different software tools for studying and manipulating finite state automata. However, the number of algebraic automata theory computational packages is more limited, and none of them included tools for the Krohn-Rhodes Theory.

The `AMORE` system is a software package for the computation of finite automata, syntactic monoids of regular languages, and (possibly star-free) regular expressions [MMP+95]. Among other functions, the program can calculate the syntactic monoid of a finite state machine and its Green $\mathcal{D}$-class picture.

`GAP` is a powerful computer algebra system for group theory [gap02]. Recent versions are extended with semigroup theoretical functionality and there is an extension package for finite state machines as well [DLM05]. This combination looks like emerging platform for dealing with automata,

and probably be the system in which the algorithms described here will be integrated.

6.2 Representational Issues

The algorithms below work with transformation semigroups, therefore it is an important question how to represent a transformation and a ts, what data structure should be used.

Transformations and Sets. Transformations are represented as mappings of the set $\mathbf{n} = \{1, \ldots, n\}$ and the 0 value is used for partial transformations (this possibility is for future extensions). No matter how an automaton is given (what symbols are used) its state set is converted to the set of the first n positive integers where n is the size of the state set, $|A|$. This inner representation is still human-readable as well since it coincides with the mathematical notation.

Transformations are stored as 1-dimensional arrays. The content of the cell with index i is the image of i. This way the multiplication of transformations can be done in linear time depending on the number of elements in A. As usually for getting fast set operations, subsets are represented as bitvectors encoding characteristic functions.

Transformation Semigroups. The efficiency of the decomposition algorithms depends largely on the way we handle semigroups. The possible representations are:

Enumeration. Exhaustive enumeration of semigroup elements.

Cayley-table. The whole multiplication table for a semigroup S. It contains $|S|$ columns and rows, one for each element. The entries of the table contains the products of the corresponding elements.

Finite presentations. Free semigroup on the input alphabet divided by the congruence of the automaton.

Generators. Generating set consisting of transformations representing the input letters.

Simple calculation shows that the first two methods are not viable options. An automaton may end up having a characteristic semigroup with n^n elements (namely the full ts) which is in the magnitude of billions already for $n = 10$. A Cayley-table is even worse as in that case it contains n^{2n} entries. Therefore we cannot have all elements at hand at a time, only a representative subset.

Finite presentations give an elegant way for defining semigroups but the decomposition proofs are not written in terms of defining relations.

The most suitable way is the last one. A generating set is a set of semigroup elements called *generators* with the property that the elements of the semigroup can be expressed as finite products of them. The generators of the characteristic semigroup of an automaton are naturally given by the transformations of the state set defined by the input symbols.

In the following algorithms it is a common question whether an element belongs to a set or not e.g. transformations in a ts. For answering these $t \in S$?-like questions in constant time we have to use hashtables [Knu98] for storing sets instead of some linear data structures (1-dimensional arrays, lists, etc.). Giving a good hashcode for a transformation is easy if it is represented as a mapping of $\{1 \dots n\}$: the sum of the images of the individual points multiplied by a fixed list of prime numbers respectively.

6.3 Trivial Implementation Using Brute Force Enumeration

Since we deal with finite structures in order to have a working implementation it is enough to fully calculate the required objects, i.e. enumerate all elements of the sets ivolved in the decomposition. Clearly this is not really clever to do, since combinatorial explosion does appear immediately. This work can be considered as an effort to replace full enumerations with more direct algorithms at the different stages of the decomposition, for the time being with partial succes. The decomposition method described here uses algorithms which are variations of the generalized breadth-first search method. Similar algorithms are called *orbit algorithms* in computational group theory [Ser03]. When we look for one particular element, or collect elements with a specific property, we use the following general method: we systematically generate elements and by using some heuristics we exclude from further generation all those not having the desired property. Since these variations of the algorithms are not restricted to calculating orbits, we call them *collector algorithms* (Alg. 1). We start from a *base set* B and by applying the *generator operation* GEN to this set we construct the set of new elements, i.e. the *result set* R, if the *terminate condition* TERM is not satisfied and there are elements to continue with, i.e. the base set is not empty. If the newly generated elements are really new ($r \notin \Delta$) and have the property Π then we collect those in a *collection set* C in case they are not contained yet. If a new element is still a candidate (CAND(r) is true) then we keep it for the next generation. Either way the element is put into the set of processed elements Δ just to keep track of what elements we have checked already. After that the result set becomes the new base set and the process is iterated.

The algorithm should collect all the elements with the desired property from a finite set. For the correctness of the general collector algorithm we

Algorithm 1: General Collector Algorithm. ADD(a, B) is a shorthand for $B \leftarrow B \cup \{a\}$ and DEL(a, B) for $B \leftarrow B \setminus \{a\}$.

```
Data  : base set B, generator operation GEN(B), terminate condition TERM(), desired property Π(e), candidate condition CAND(e)
Result: collection set C, Π(c) holds ∀c ∈ C
Δ ← ∅ ;
while (not TERM()) and (B ≠ ∅) do
    R ← GEN(B);
    foreach r ∈ R do
        if r ∉ Δ then
            if Π(r) then ADD(r, C);
            if not(CAND(r)) then DEL(r,R) ;
            ADD(r, Δ);
        end
        else DEL(r, R);
    end
    B ← R;
end
```

need the following requirements:

- an element, which does not satisfy the candidate condition can be excluded from the generator set, therefore it cannot reappear in the base set
- all elements with the desired property are accessible from the generator set by the generator operation

Regarding the space complexity of the collector algorithm in general we can only say that it is bounded by the cardinality of the set of all elements that can be generated from the base set by the generator operator. The time complexity has at least the same bound but it depends on how effective the generator operator is , i.e. how many times it generates elements already processed.

6.4 Examples

6.4.1 Generating Images

Generating the elements of $\mathcal{I}$ is a good example, where the base set is the state set, the generator operation is the multiplication with the generator transformation(s) of the ts. The required property is being an image (which

we have in this case by default), the candidate condition is that the image should be new (not in the collection yet), and the algorithm terminates when the base set is empty.

Algorithm 2: Image Enumerator Algorithm. Here we do not need a separate set for the processed elements since we need all of the images.

```
Data  : base set 𝓑 = {A}, GEN(A): action of a generator set G of S,
        TERM():constant false, Π(P),CAND (P): constant true, i.e.
        just being a new element
Result: 𝓒: the set of images of S
while 𝓑 ≠ ∅ do
    𝓡 ← GEN(B);
    foreach P ∈ 𝓡 do
        if P ∉ 𝓒 then ADD(P,𝓒);
        else DEL(P,𝓡);
    end
    𝓑 ← 𝓡;
end
```

6.4.2 Deciding Subduction Relation

A slightly different problem is deciding the subduction relation, whether $P \leq Q$, i.e. looking for a witness. The base set is G, the generator set of the characteristic ts, the generator operation is simply multiplication by generators, the candidate condition is $\mathrm{CAND}(R) = |P| \leq |Q \cdot R|$, the desired property is $\Pi(R) = |P| \subseteq |Q \cdot R|$, which is exactly the subduction relation. The termination condition is $\mathrm{TERM}() = C \neq \emptyset$, thus we stop whenever a witness is found, so the relation holds, or when the base set is empty, thus the relation does not hold.

6.4.3 Holonomy Components

One possible and natural way to obtain a holonomy group H_Q is first collecting those elements of S which permute Q (called its *permutators*) then taking a homomorphic image of this collected set by collapsing transformations having the same effect on B_Q (a permutator can permute elements within a tile, or two permutators might be different regarding $A \setminus Q$ but doing the same with Q). But if we make the homomorphic mapping while collecting then we may avoid elements that are collapsed in the homomorphic image. We can stop collecting if the maximum of the sizes of noncollapsing subsets of A in the base set's transformations is smaller than $|Q|$. This reduces the search space substantially if $|Q|$ is relatively big. The time and space complexity is $O(n^n - (n-k)^{(n-k)})$ where $k = |Q|$, but this might be

Algorithm 3: Subduction Relation Algorithm

```
Data   : base set G, GEN(G: generating elements of S), TERM: C ≠
         ∅ i.e. a witness is found, Π(w): w is a witness, CAND(r):
         |P| ≤ |Q · R|
Result: C containing at least one witness
Δ ← ∅;
while (C = ∅ ) and (B ≠ ∅) do
    R ← GEN(B);
    foreach r ∈ R do
        if r ∉ Δ then
            if P ⊆ Q · r then ADD(r, C);
            if |P| > |Q · r| then DEL(r,R);
            ADD(r, Δ);
        end
        else DEL(r, R);
    end
    B ← R;
end
```

close to enumerating all elements for small k. In a special case we can also stop collecting if the symmetric group is found (once some holonomy group elements are found we can use them as a generator set and check the order of the group).

6.5 Visualization

"We must see the matter at once, at one glance, and not by a process of reasoning, at least to a certain degree."
Blaise Pascal, Pensees I/1.

Our conceptual system is firmly grounded in our spatial sensory system. Some research even say that our logical (and mathematical) thinking is deeply rooted in spatial reasoning [LN00]. We would not like to continue here the discussion about the nature of mathematics, but we would like to emphasize the importance of good diagrams in mathematical research. For instance, the whole tiling picture can be shown on a graph, where the layout enables us to grasp the full structure, not just the pieces delivered by formulas.

Our sowftware toolkit outputs the description of the graphs to be displayed and the actual layout is rendered by an external package. For this purpose we use the widely known and used software package called `GraphViz` [EGK+03].

Algorithm 4: Holonomy Group Algorithm

```
Data  : a subset of A: Q, base set G, GEN(G: generating elements
        of S), TERM: C is the symmetric group on B_Q, Π(w): w
        permutes B_Q, CAND(r): BiggestNonCollapsingSet(r) ≥ |Q|
Result: C = H_Q, the holonomy group of Q
while (|C| ≠ |B_Q|!) and (B ≠ ∅) do
    R ← GEN(B);
    foreach r ∈ R do
        if r ∉ P then
            if B_Q · r = B_Q then ADD(r, C);
            if BiggestNonCollapsingSet(r) < |Q| then DEL(r, R);
            ADD(r, P);
        end
        else DEL(r,R);
    end
    B ← R;
end
```

6.6 Summary

Some decisioins about the low level details of a computational implementations for Krohn-Rhodes Theory (in particular and computational semigroup theory in general) had to be made before proceeding to the more interesting computational problems. The actual chosen representations come from the programmers' common sense. The problem here in semigroup theory can be described very shortly: semigroups tend to have extremely many elements.

The collector algorithms based on the idea of a brute force search, but they suffice for some specific problems like the image generation task and the problem of deciding the subduction relation. They can be applied whenever the generator operation is not redundant and the termination condition is "quick", in the sense that the probability of terminating in the beginning of the search is quite high. One might ask for the exact time and space complexity of these algorithms, but these exponential algorithms are likely to be replaced by more efficient algorithms when our knowledge about these decompositions advance.

But the brute force algorithm is not sufficient for the initial exploration decompositions in the case of constructing the holonomy components. Therefore the whole next chapter is devoted to this problem.

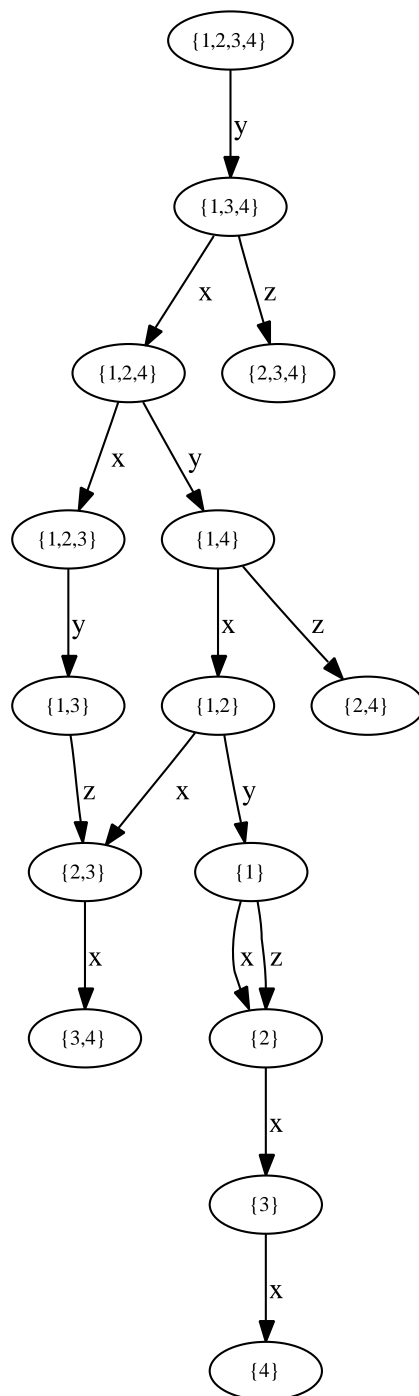


Figure 6.1: Generating the image set for $\mathcal{T}_4$, the full transformation semigroup on 4 points. The input symbols correspond to the following transformations: a transposition $x = (2\ 3\ 4\ 1)$, a collapser $y = (1\ 1\ 3\ 4)$, and a cyclic permutation $z = (2\ 3\ 4\ 1)$. The arrows denote the actions of the input symbols. Note that this is not the full image relation graph (but a subgraph of it), and this shows how the Image Enumerator Algorithm works.

Chapter 7

Constructing Holonomy Components

When constructing the holonomy components we need to find certain subgroups (or generators of subgroups) of the characteristic semigroup. This also can be done by using the collector algorithms, but since transformation semigroups can have so many elements we immediately bump into complexity issues. Some tricks can be used (for instance recognizing the generator set of the symmetric groups), but this problem demands a more systematic solution. Here we propose two methods that provide improved solutions. First we define the problem precisely, then show two different solutions.

7.1 The Problem

Let (A, S) be the corresponding ts of a given an automaton $\mathcal{A} = (A, X, \delta)$, and let Q be an arbitrary nonsingleton element of $\mathcal{I}$, which is a subset of the state set A. Then, we would like to construct the permutation group (Q, G_Q) induced by the elements of S^I, where G_Q is a maximal permutation group on Q, i.e. no subgroup of S^I contains G_Q properly, when it is restricted to Q. We call G_Q the permutator group of Q. Moreover, it is also a question to decide whether G_Q is trivial or not.

The ultimate goal is to have H_Q, the holonomy group of Q, which is a subset of G_Q, the set of transformations that permute B_Q.

Obviously, the task of finding H_Q can be accomplished in every case due to the finiteness of the automaton. We only need the characteristic semigroup for checking for its elements systematically whether they permute Q or not. But this method is not satisfactory for two reasons:

Computational efficiency. The characteristic semigroup can be big even for simple automata (in the worst case n^n for T_n), therefore the enumeration of all elements is not a usable approach for a computational

method. Different techniques (checking the size of images, stopping when a symmetric group is found) can be applied in order to reduce the search space to some extent.

Lack of a small generator set for H. Due to the advanced group theoretical algorithms [Ser03] it is enough to have a small set of generators of a group instead of the explicit set of all group elements, but the enumeration method gives the full permutation group.

In sum, the main fault of the enumeration method is that it is a "blind" search rather than a direct way of constructing H_Q. In the following we describe a method using words on the alphabet of the generating input symbols, which overcomes the shortcomings of the plain enumerative method.

7.1.1 Examples

The difference between H_Q and G_Q

Example 7.1 *Let $p_1 = (2\ 3\ 1\ 5\ 6\ 4)$ and $p_2 = (4\ 5\ 6\ 1\ 2\ 3)$ be permutations of the set $Q = \{1,2,3,4,5,6\}$, and $B_Q = \{\{1,2,3\},\{4,5,6\}\}$ a tiling of Q.*

Clearly both p_1 and p_2 are nontrivial permutations of Q, but acting on B_Q p_2 gives the identity permutation in H_Q. Those permutations in G_Q, which move elements inside the tiles only, they fall into the identity of H_Q by the surjective homomorphism from G_Q to H_Q.

7.2 Word Based Construction Method

Intuitively permutations are connected to cycles in the state transition graph, so in order to identify permutation subgroups of the characteristic we need to check cycles of automata. This intuitive idea is more or less right, but loosely speaking, not every cycle corresponds to a permutation group element. Therefore we have to clarify the notions of different cycles: the *graphical cycle*, which looks like and may be a permutation, and the *algebraic cycle* which really is a permutation. The distinction is made by some properties of the labelling word.

Based on these notions we give simple classification of automata and show that the construction of the holonomy components can be done by examining the cycle structure of certain derived automata.

7.2.1 Cycles in Automata

Definition 7.2 *A* graphical cycle *in an automaton (A, X, δ) is a cycle in its state transition digraph together with a word $w \in X^+$, i.e. a sequence of states $a_1, \dots, a_n$ $n \geq 2$, where the states in the sequence are pairwise*

distinct except $a_1 = a_n$*, and* $w = x_1 \dots x_{n-1}$*,* $x_i \in X$ *such that* $a_i \cdot x_i = a_{i+1}$ *for all* $1 \leq i \leq n-1$*. The word* $w = x_1 \dots x_{n-1}$ *is called the* label *of the cycle.*

Since $n \geq 2$ a loop edge is not a graphical cycle, and also, since $a_i \neq a_{i+1}$ within a graphical cycle, loop edges are not allowed.

Definition 7.3 *An* algebraic cycle *in an automaton* $\mathcal{A} = (A, X, \delta)$ *is a permutation group* $(\{a_1, \dots, a_n\}, \langle w \rangle)$ *for which* $a_i = a_j \Rightarrow i = j$*,* $n > 1$*, and* w *is a word in* X^+ *such that* $a_i \cdot w = a_{i+1}$ *for all* $1 \leq i < n$*, and* $a_n \cdot w = a_1$*.*

The word w generates a cyclic group which acts faithfully on $\{a_1, \dots, a_n\}$ by permutations. (Of course $\langle w \rangle$ might not act by permutations on A.) Obviously w^n is the identity element. Moreover, n being greater than 1 excludes trivial one-element groups. Note that loops are not generally algebraic cycles. The *generator* of the algebraic cycle is w, and its label is w^n.

7.2.2 Graphically Cycle-Free Automata

Definition 7.4 *An automaton is* graphically cycle-free *if it does not have any graphical cycle.*

The very simple structure of graphically cycle-free automata is reflected in their subduction pictures in the following way:

Lemma 7.5 (A, S) *is graphically cycle-free iff on every height level in each subduction relation equivalence class there is only one element.*

Proof: Let $P, Q \in \mathcal{I}$ and $P \equiv Q$ but $P \neq Q$. Since P, Q are finite $|P| = |Q|$. Clearly by finiteness there is at least one $x \in Q$ such that $x \notin P \cap Q$, otherwise P, Q would be the same. Due to the equivalence of P and Q we have $s, t \in S$ bijective mappings such that $P = Q \cdot s$ and $Q = P \cdot t$ and thus $(st)^n$ is the identity on Q for some $n > 0$, by the finiteness of P, Q. Since $x \cdot s = x' \neq x$ while $x \cdot (st)^n = x$, there must be a graphical cycle.

Conversely, a graphical cycle ensures the existence of an equivalence class with at least two elements at height zero. □

Another way to think about the proof of this lemma is to recognize that for the singleton subsets of the state set (at height zero) the equivalence classes are exactly the strongly connected components of the automaton's state transition graph.

This result can be exploited in the decomposition algorithm since if the equivalence classes are detected to all be singleton classes, then there is no need to look for holonomy groups at all and the holonomy identity-reset ts's can be built immediately.

7.2.3 Algebraically Cycle-Free Automata

It is a well-known result of algebraic automata theory that the star-free rational languages are recognized by exactly those automata whose characteristic monoid is aperiodic (having no nontrivial subgroup) [Sch65]. It is also known that deciding aperiodicity for a finite automaton is PSPACE-complete [CH91]. We are interested in this problem for certain derived automata that arise naturally in the holonomy decomposition.

Intuitively one might expect that the state transition graph of an aperiodic automaton contains no cycles at all, but this is not true in general: there might be graphical cycles in it, while remaining aperiodic (see Fig 7.1). But with another type of cycles the notion of aperiodicity can be expressed.

Definition 7.6 *An automaton* $\mathcal{A} = (A, X, \delta)$ *is* algebraically cycle-free *if it does not have any algebraic cycle.*

The property of algebraic cycle-freeness is tied up with the primitivity of words, which act on some states as the identity.

Lemma 7.7 *An automaton* $\mathcal{A} = (A, X, \delta)$ *is algebraically cycle-free iff for all states* $a \in A$ *and for all words* $w \in X^+$ *such that* $a \cdot w = a$ *, one of the following statements holds.*

1. w *is primitive.*
2. w *is not primitive but has primitive root* $u \in X^+$*, i.e.* $w = u^n$*, and* $a \cdot u = a$*.*

Proof: If w is primitive, then we are done. Otherwise $w = u^n$ where u is primitive. Let's suppose indirectly that $a \cdot u \neq a$. Let k be the least integer that $a \cdot u^k = a$ $(1 < k \leq n)$. Then $(\{a, a \cdot u, \ldots, a \cdot u^{k-1}\}, \langle u \rangle)$ is a cyclic permutation group (with at least two elements), therefore we have an algebraic cycle, contradicting our assumptions.

The converse is obvious due to the fact that a trivial permutation group does not constitute an algebraic cycle, and the conditions $1 - 2$ allow only trivial permutation groups. □

Remark 7.8 *Obviously Lemma 7.7 holds even if* $a \cdot z \neq a$ *for some left factor* z *of* w*.*

It is clear that in the absence of graphical cycles there cannot be any algebraic cycle. Thus,

Proposition 7.9 *If an automaton is graphically cycle-free then it is algebraically cycle-free.*

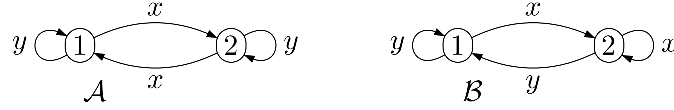


Figure 7.1: Automaton $\mathcal{A}$ has an algebraic cycle $(\{1,2\}, \langle a \rangle)$. Automaton $\mathcal{B}$ has graphical cycles ab, ba, but they are labeled with primitive words.

Now we show that aperiodic automata are exactly the algebraically (not the graphically) cycle-free ones.

Theorem 7.10 *The following are equivalent for an automaton $\mathcal{A} = (A, X, \delta)$ with corresponding transformation semigroup (A, S):*

1. *$\mathcal{A}$ is algebraically cycle-free.*
2. *S is aperiodic.*
3. *Holonomy groups are trivial for (A, S).*

Proof: (1) $\Rightarrow$ (2): Suppose S is not aperiodic, then we have a cyclic group $\langle v \rangle$ in S of order $n \geq 2$, where $v \in X^+$ is a word representing the generator. Thus v^n is the identity of the cyclic group, $v \equiv v^{n+1}$ and $v \not\equiv v^2$. Therefore $\exists a$ such that $a \cdot v \neq a \cdot v^2$ and $a \cdot v = a \cdot v^{n+1}$. Let $a' = a \cdot v$, thus $a' \cdot v^n = a'$ and since $\mathcal{A}$ is algebraically cycle-free we can apply Lemma 7.7: let $u = \sqrt{v^n} = \sqrt{v}$, then we have $a' \cdot u = a'$, $a' \cdot v = a'$ and finally $a \cdot v^2 = a \cdot v$, which is a contradiction.

(2) $\Rightarrow$ (1): For the converse we use again an indirect proof: Suppose there is an algebraic cycle, i.e. $(\{a_1, \ldots, a_n\}, \langle w \rangle)$ is a permutation group with $a_i \in A, w \in X^+$ and $n > 1$. Therefore $\mathbb{Z}_n$, the cyclic group with n elements, divides S. This cannot happen when S is aperiodic.

(2) $\Leftrightarrow$ (3): The components of the holonomy decomposition are all divisors of the original semigroup, thus aperiodic semigroups have only trivial holonomy groups, and wreath products and divisors of aperiodic transformation semigroups are aperiodic. □

Corollary 7.11 *An automaton $\mathcal{A} = (A, X, \delta)$ is aperiodic if and only if*

$$\forall a \in A,\, w \in X^+,\, x \cdot w = a \Rightarrow a \cdot \sqrt{w} = a.$$

The distinction between algebraically cycle-free aperiodic and nonaperiodic automata is rather subtle. Two automata having the same state-transition graphs regarding their connectivity might belong to different classes depending on how the input symbols act on the state set (Fig. 7.1).

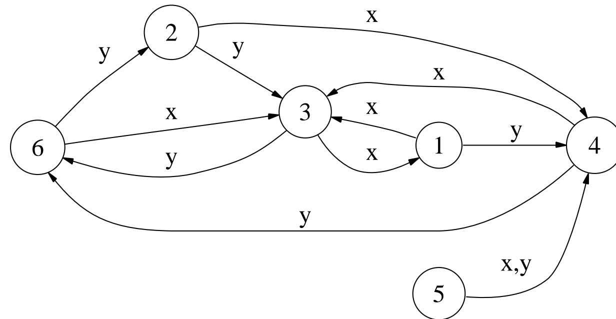


Figure 7.2: An automaton $\mathcal{A}$ with state set $A = \{1, 2, 3, 4, 5, 6\}$ and alphabet $\{x, y\}$, where x and y are transformations with $x = (3\ 4\ 1\ 3\ 4\ 3)$, $y = (4\ 3\ 6\ 6\ 4\ 2)$.

7.2.4 Non-Aperiodic Automata

A main concern of the holonomy decomposition is to find the nontrivial holonomy groups. Fortunately the tiling picture provides tools for locating the elements of $\mathcal{I}$ for which there exist nontrivial holonomy groups.

Lemma 7.12 *For an element Q of $\mathcal{I}$ in the tiling picture of (A, S) if there is a nontrivial holonomy group H_Q, then in its set of tiles B_Q there are at least two distinct tiles t_1, t_2 such that $t_1 \equiv t_2$.*

Proof: H_Q being nontrivial means that there are some pair(s) of tiles for which there are transformations permuting them and thus they are mutually subduction related. □

The converse is not generally true as we can see in the example of an automaton (Fig 7.2) with tiling picture (Fig 7.3). For a trivial H_Q the set of tiles B_Q may contain distinct equivalent tiles, see Fig 7.4. In order to determine whether we have a nontrivial holonomy group for a $Q \in \mathcal{I}$ we define an extended automaton and examine its cycle structure. Denote the equivalence classes of subduction relation by E_1 to E_N.

Lemma 7.13 *If $P \in E_i$ and for some $s \in S$, $P \cdot s = Q$ such that $Q \notin E_i$ (leaving the equivalence class) then there is no transformation $t \in S$ such that $Q \cdot t \in E_i$ (no way back to the original equivalence class).*

Proof: Suppose there is such a t that $Q = P \cdot s$ and $P' = Q \cdot t$ with $P \equiv P'$. Due to the equivalence we have $P = P' \cdot s''$ for some $s'' \in S$, therefore $Q \cdot (ts'') = P' \cdot s'' = P$, thus $Q \equiv P$, which contradicts the original assumption that we leave the equivalence class of P. □

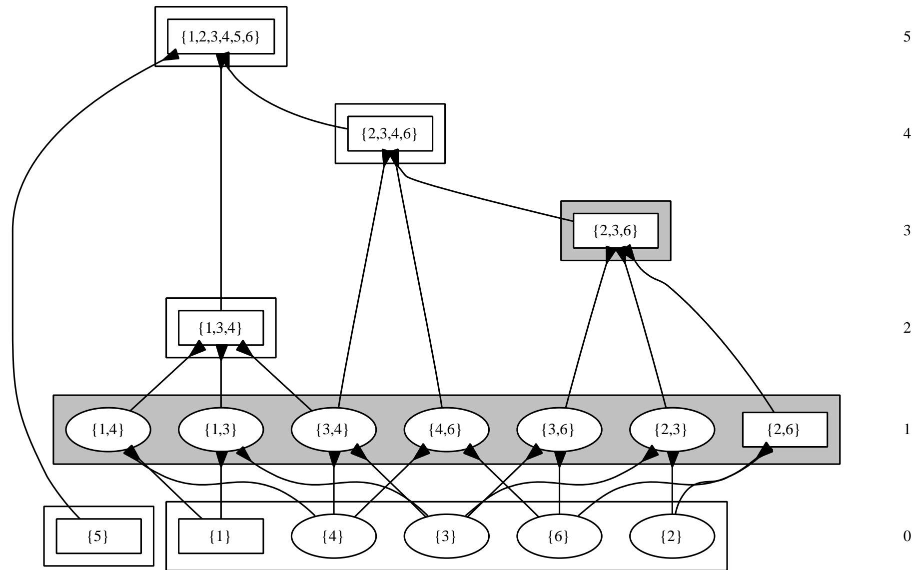


Figure 7.3: The tiling picture of automaton $\mathcal{A}$ in Fig. 7.2. The equivalence classes are denoted by boxes. Equivalence classes with elements having nontrivial holonomy groups are shaded. Arrows ending in plugs denote the 'tile of' relation.

Let's define E_Q as the union of equivalence classes which contain at least one tile of $Q \in \mathcal{I}$. Formally: $E_Q = \bigcup_{E_i \cap B_Q \neq \emptyset} E_i$. Then the *tile automaton* of Q is defined as $\mathcal{A}_Q = (E_Q \cup \{\varsigma\}, X, \gamma)$, where ς is a sink state, the input alphabet X is the same as the original automaton's, and γ is the natural extension of δ to act on subsets of A providing that if the image is not in some E_i then it is ς. This way ς represents going to another equivalence class not contained in E_Q, but according to Lemma 7.13 this can be represented as a sink since there is no way to come back.

The equivalence classes in E_Q form strongly connected components in $\mathcal{A}_Q$. When determining the nontriviality of H_Q we look for algebraic cycles in these components. We look not simply for independent algebraic cycles in each component as a word of a cycle might not permute the tile elements in another component, but for parallel algebraic cycles. This way we can recast the characterization of a holonomy group element in terms of algebraic cycles. More formally:

Proposition 7.14 *H_Q is nontrivial iff there exists a word $w \in A^+$ and B_Q can be partitioned into $\{T_1, \dots, T_k\}$ subsets such that either*

1. *T_i consists of exactly one tile and $T_i \cdot w = T_i$, or*

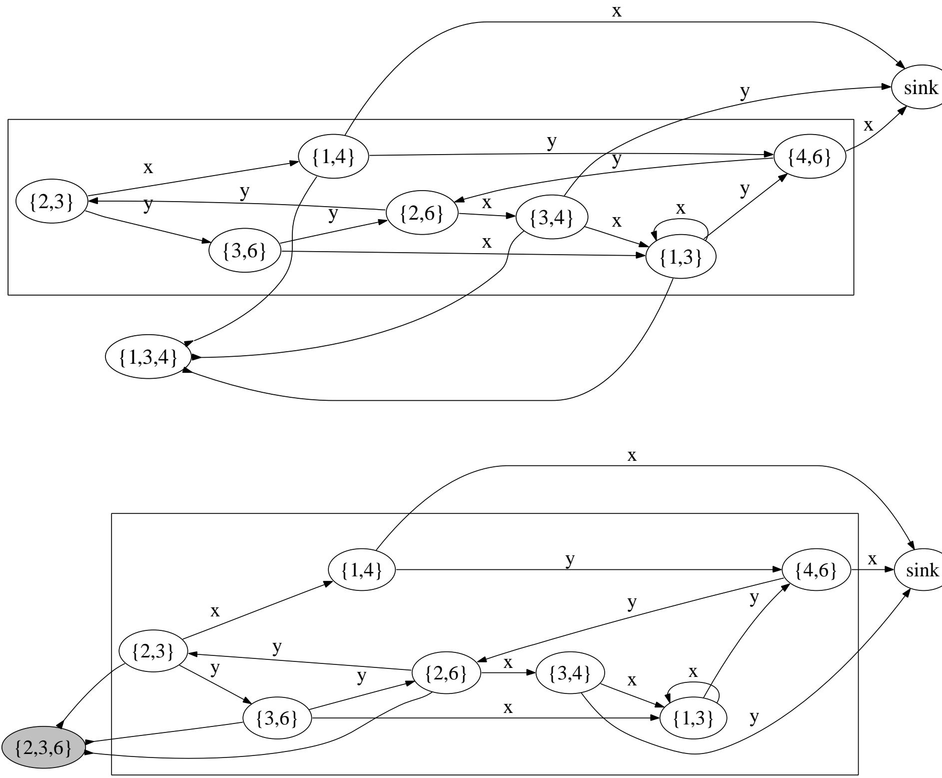


Figure 7.4: Two tile automata of automaton $\mathcal{A}$ in Fig. 7.2. $\mathcal{A}_{\{1,3,4\}}$ is trivial, while $\mathcal{A}_{\{2,3,6\}}$ is nontrivial with generator word y.

2. $T_i \cdot \langle w \rangle \subseteq B_Q \cap E_j$ *for some* $1 \leq j \leq N$, *and* $(T_i \cdot \langle w \rangle, \langle w \rangle)$ *is an algebraic cycle in* $\mathcal{A}_Q$

holds for all T_i, $1 \leq i \leq k$, *and* (2) *must hold for at least one* T_i.

In short the proposition characterizes when the transformation induced by w nontrivially permutes B_Q. This transformation is clearly a nontrivial holonomy group element. From Lemma 7.13, $T_i \cdot w^n \in (B_Q \cap E_j)$ follows for any $n \geq 0$. Therefore the algebraic cycles contained in B_Q generated by w are all disjoint. If all intersections $(B_Q \cap E_j)$ are singletons, or none of them contains an algebraic cycle then H_Q is trivial. This fact can be exploited in efficient decomposition algorithms of the holonomy decomposition by excluding cases where the construction of the holonomy group should not be attempted.

We describe an algorithm for constructing the maximal group permuting an arbitrary given subset of a finite state automaton induced by the

input symbols, and characterize its computational complexity. This problem naturally arises in the computational implementation of the holonomy decomposition for the Krohn-Rhodes Theory.

7.2.5 The Algorithm

We described the problem on two different levels: finding permutators and finding the holonomy group elements. In order to avoid the notational burden we present the solution for the permutator problem. It is easy to translate into the holonomy group problem (but not the opposite direction). Once we know (Q, H) then (B_Q, H_Q) is obtained by making (B_Q, H) be faithful, as in Example 7.1.

This algorithm is yet another example of collector algorithms, so the idea is simple again. We generate words and collect hose that induce permutations on Q. But the details are a bit more complicated.

Generator Operation

We can just generate words starting from the empty word and applying input symbols systematcially. But we know something about the form of the possible candidate words. Since the words labeling transformations in G_Q should permute Q, the corresponding paths start from and end in Q. So we define the following sets of input symbols:

$$Q_{\text{out}} = \{x \in X \mid \exists a \in Q, b \in A \ \text{ such that } a \cdot x = b\}$$

$$Q_{\text{in}} = \{x \in X \mid \exists a \in Q, b \in A \ \text{ such that } b \cdot x = a\}$$

Note that the names might be a bit misleading, since an incoming edge might come from Q itself, and the outgoing might stay in Q. In the next section it will become clear why we choose these names.

The generation of words has three stages:

1. Single letter words in $Q_{in} \cap Q_{out}$.
2. Two letter words in $Q_{in} \times Q_{out}$.
3. Words in the form $x_i w x_o$, where $x_i \in Q_{in}$, $x_o \in Q_{out}$, $w \in X^*$ generated systematically in alphabetical order.

This way we do not need to check words that clearly cannot permute Q.

Candidate Property

We know that the paths corresponding to the permutations of Q start and end in Q. But how freely can they go around meanwhile? We can go "as far" as we like until we can come back eventually, which – in terms of the

state transition graph – means that we cannot leave the strongly connected components. The orbit of one particular point a should stay within the strongly connected components of a denoted by C_a. $C_Q = \bigcup_{a \in Q} C_a$, is the union of strongly connected components of the states in Q. Finding the strongly connected components of a digraph (here of a state transition graph) can be done in linear time by acyclic ordering (topological sorting) of vertices using depth-first search [BJG02].

A word w represents a permutation of Q if $Q \cdot w = Q$. Once a word w is found to be a permutation, we can stop extending it. This can be shown by a simple argument: w is *minimal* in the sense that no proper left factor of it is a permutation. Suppose wv is the next permutation in the continuation of w (no proper left factor is a permutation, except w). Then, either $w = v$, or v itself is a permutation. In the first case we clearly do not loose anything by stooping at w. The same is true for the second case, since if v is different then it is found independently as another branch of the search-tree.

A particular permutation (and transformations in general) can be expressed by many different words. We do not need to keep track of all those words but one. Therefore we define an equivalence relation on words by:

$$w \equiv_Q v \text{ iff } a \cdot w = a \cdot v \quad \forall a \in C_Q,$$

and it is enough to deal only with the equivalence classes $X^+ / \equiv_Q$ of words. A class is represented by one of its elements, namely the first word found by the breadth-first search. Obviously, we can safely ignore what permutations do on $A \setminus C_Q$.

In sum, for each newly generated word w we check whether the following properties are valid:

- a word equivalent (realizing the same transformation when restricted to C_Q) to w is already processed,
- w is a permutation of Q (this is the desired property),
- $a \cdot w \notin C_a$ for any $a \in Q$ (leaves the strongly connected component),
- $|Q \cdot w| < |Q|$ (some states in Q are collapsed).

If any of these conditions is true, then we stop extending w. Otherwise, if none of the above conditions are satisfied then we continue extending w. Clearly, the generations terminates, since we collect the transformations (not the possibly infinitely many word representations of them) and they are finitely many.

7.2.6 Examples

Q_{in} and Q_{out}

Consider Figure 7.4 again. $B_{\{1,3,4\}}$ is $\{\{1,3\},\{1,4\},\{3,4\}\}$. $\{1,3,4\}_{in} = \{x\}$, $\{1,3,4\}_{out} = \{x,y\}$. In the first case we have some search space reduc-

tion, while in the case of outgoing edges we gain nothing as the set of input symbols is exactly $\{x, y\}$.

$B_{\{2,3,6\}}$ is $\{\{2,3\},\{2,6\},\{3,6\}\}$. $\{2,3,6\}_{in} = \{y\}$, $\{2,3,6\}_{out} = \{x,y\}$. In the first case again there is some search space reduction. And the very first step in the generation of words finds a generator: $\{2,3,6\}_{in} \cap \{2,3,6\}_{out} = \{y\}$.

The Good and the Bad

Again, we would like to use the full transformation semigroup for testing our method. The main concern now is the number of generators provided by the algorithm. Recall that in the decomposition of the full ts we have symmetric groups (plus constant maps) on all levels. Therefore the components can be generated only by two permutations, a cycle and a transposition. Decomposing $\mathcal{T}_6$ gives the following number of generators:

Level	#Generators	Order of holonomy group
1	**2**	2
2	**6**	6
3	**24**	24
4	40	120
5	2	720

Boldface numbers indicate cases where the generator set equals the generated group. On the top level it is the minimal number of generating elements and this efficiency is due to the fact it is 'easy to fall down from the peak', i.e. there are not many elements in the equivalence class to wander around. But when the tiling picture gets wider the algorithm becomes very inefficient: it enumerates all elements of the generated group.

Curious readers might want to check the output of the algorithm on the fourth level. It is a challenging exercise to track down the generator words (listed on Fig. 7.6) on Fig. 7.5.

7.3 Dependency Function Based

The real machinery of the cascaded automaton hides in the dependency functions. We tried to emphasize this insight before and now we present something more convincing: a method for constructing holonomy components by using the values (component actions) of the dependency functions. We use the algorithm for lifting the transformations for finding the holonomy group generators.

The idea is simple and comes from the following origins:

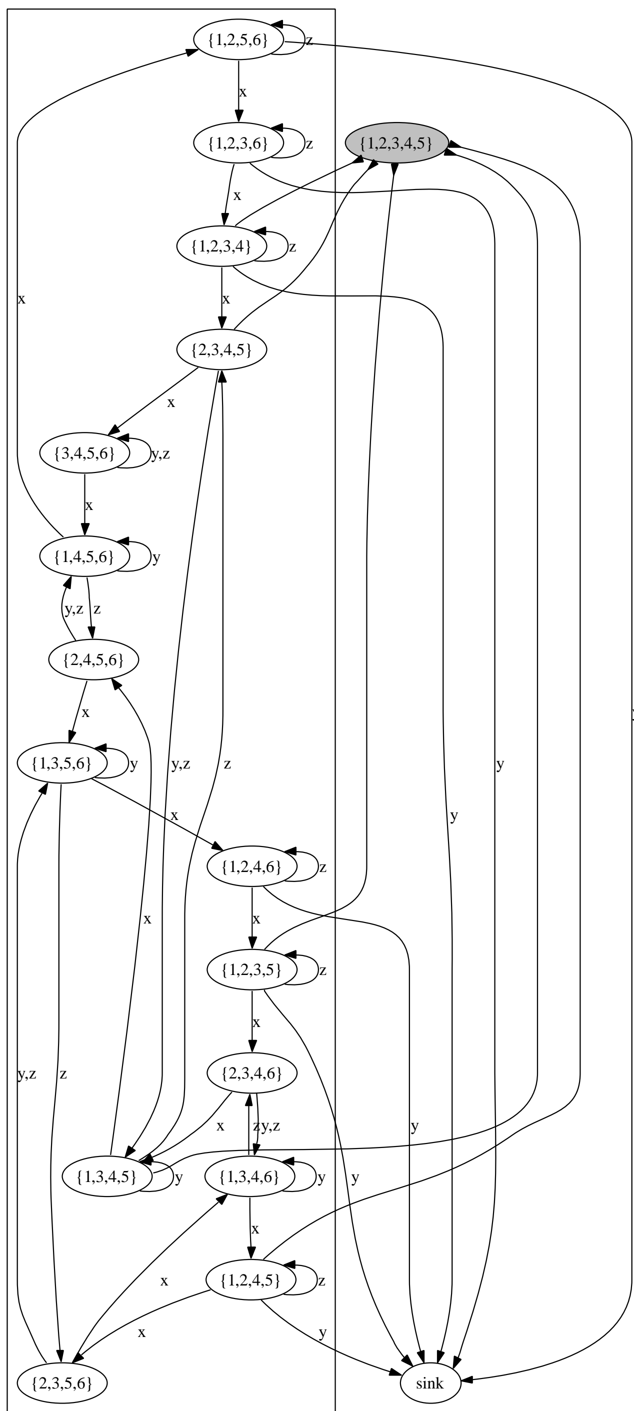


Figure 7.5: The tile automaton of of the fourth level's component of the holonomy decomposition of $\mathcal{T}_6$. The generators are: $x = (2\ 3\ 4\ 5\ 6\ 1)$, $y = (1\ 1\ 3\ 4\ 5\ 6)$, $z = (2\ 1\ 3\ 4\ 5\ 6)$.

Word	in G_Q	in H_Q
z	(2 1 3 4 5 6)	(4 2 3 1 5)
$xyxxxx$	(5 1 2 3 4 5)	(4 5 1 2 3)
$xxxxxx$	(1 2 3 4 5 6)	(1 2 3 4 5)
$xzxxzxx$	(5 1 2 4 3 6)	(4 3 1 2 5)
$xzxxxzx$	(5 1 3 2 4 6)	(4 5 3 2 1)
$xxxzxxx$	(1 2 3 5 4 6)	(1 5 3 4 2)
$xxxxzxx$	(1 2 4 3 5 6)	(1 2 5 4 3)
$xyxzxxx$	(4 1 2 3 5 4)	(4 2 1 5 3)
$xxxxxzx$	(1 3 2 4 5 6)	(3 2 1 4 5)
$xyxzxxzx$	(4 1 3 2 5 4)	(4 2 3 5 1)
$xyxzxzxx$	(3 1 2 4 5 3)	(4 2 1 3 5)
$xxxzxxzx$	(1 3 2 5 4 6)	(3 5 1 4 2)
$xxxxzxzx$	(1 3 4 2 5 6)	(3 2 5 4 1)
$xxzxxxxx$	(2 3 4 5 1 6)	(3 4 5 1 2)
$xzxxzxzx$	(5 1 3 4 2 6)	(4 1 3 2 5)
$xxxzxzxx$	(1 2 4 5 3 6)	(1 3 5 4 2)
$xxxzxzxzx$	(1 3 4 5 2 6)	(3 1 5 4 2)
$xxzxxxxzx$	(3 2 4 5 1 6)	(1 4 5 3 2)
$xxzxxzxxx$	(2 3 5 4 1 6)	(3 4 2 1 5)
$xxzxxxzxx$	(2 4 3 5 1 6)	(5 4 3 1 2)
$xxzxxzxzxx$	(2 4 5 3 1 6)	(5 4 2 1 3)
$xxzxxzxxzx$	(3 2 5 4 1 6)	(1 4 2 3 5)
$xxzxxxzxzx$	(3 4 2 5 1 6)	(5 4 1 3 2)
$xxzxzxxxxx$	(3 4 5 1 2 6)	(5 1 2 3 4)
$xxzxzxxzxxx$	(3 5 4 1 2 6)	(2 1 5 3 4)
$xxzxxzxzxzx$	(3 4 5 2 1 6)	(5 4 2 3 1)
$xxzxzxxxzxx$	(4 3 5 1 2 6)	(3 1 2 5 4)
$xxzxzxxxxzx$	(2 4 5 1 3 6)	(5 3 2 1 4)
$xxzxzxzxxxxx$	(4 5 1 2 3 6)	(2 3 4 5 1)
$xxzxzxxzxzxx$	(4 5 3 1 2 6)	(2 1 3 5 4)
$xxzxzxxxzxzx$	(4 2 5 1 3 6)	(1 3 2 5 4)
$xxzxzxxzxxzx$	(2 5 4 1 3 6)	(2 3 5 1 4)
$xxzxzxzxxxxzx$	(4 5 1 3 2 6)	(2 1 4 5 3)
$xxzxzxzxxxzxx$	(3 5 1 2 4 6)	(2 5 4 3 1)
$xxzxzxzxxzxxx$	(5 4 1 2 3 6)	(5 3 4 2 1)
$xxzxzxxzxzxzx$	(4 5 2 1 3 6)	(2 3 1 5 4)
$xxzxzxzxxzxxzx$	(5 4 1 3 2 6)	(5 1 4 2 3)
$xxzxzxzxxzxzxx$	(5 3 1 2 4 6)	(3 5 4 2 1)
$xxzxzxzxxxzxzx$	(2 5 1 3 4 6)	(2 5 4 1 3)
$xxzxzxzxxzxzxzx$	(5 2 1 3 4 6)	(1 5 4 2 3)

Figure 7.6: The generator words for S_5 in the decomposition of $\mathcal{T}_6$ produced by the word based construction method.

Lifting generators only. For lifting the transformation semigroup to the wreath product semigroup it is enough to map the generators only (see division in Section 2.1.5). This implies that the dependency functions of the generators contain all information about the generators of the holonomy components, thus they encode the whole wreath product ts.

Holonomy permutations are moves within equivalence classes. With the help of $\overleftarrow{m}_P$ and $\overrightarrow{m}_P$ we can move between an arbitrary element and the representative of a subduction equivalence class. By concatenating these maps we can move from any element to any other element within the equivalence class. Thus having certain set of these maps might draw the generator set for holonomy components, since a holonomy permutation is built up from synchronized movements in some equivalence classes, as showed in Section 7.2.4.

'Cheating' gives component actions. For lifting the original ts in the mathematical proof of the Holonomy Decomposition Theorem (Section 5.6.1) we do some kind of cheating: we do not give the lift in its full detail (in a small set of formulas or as a lookup table serving the needs of a practical minded computer scientist), we just describe a method for mimicking the lifted transformation in a particular situation (i.e. a given cascaded state), for giving the component actions by combining $\overleftarrow{m}_P$ and $\overrightarrow{m}_P$ maps. By knowing the original transformation we tell for a particular cascaded state what the transformation lift would do in a particular situation, which are exactly the component actions.

7.3.1 The Algorithm

Since we cannot give a nice formula for the dependency functions, we handle these functions as lookup tables. The tables map *upper fragments* $(B_i, \dots, B_h),\ 1 \le i < h$ of cascaded states, the elements in $\mathcal{B}_1 \times \cdots \times \mathcal{B}_h$ to holonomy actions (constants and permutations). To make these functions fully defined for those upper fragments not appearing in the lookup table we define their values to be the identity map. The steps for getting the holonomy generators are the following:

1. Generating $\Lambda(A)$, i.e. enumerating all tile chains. Clearly, in terms of efficiency this is the weakest part of the algorithm, since there can be many tile chains. For the time being we do not know what subset of the tile chains is needed for defining the dependencies.

2. Applying each generator lift to $\Lambda(A)$ by using the method for lifting the transformations described in Section 5.6.1. Whenever we get a nontrivial holonomy group action we record the arguments of the dependency function and its value. A nontrivial action is a constant map or a permutation which does not act as an identity on the tiles.

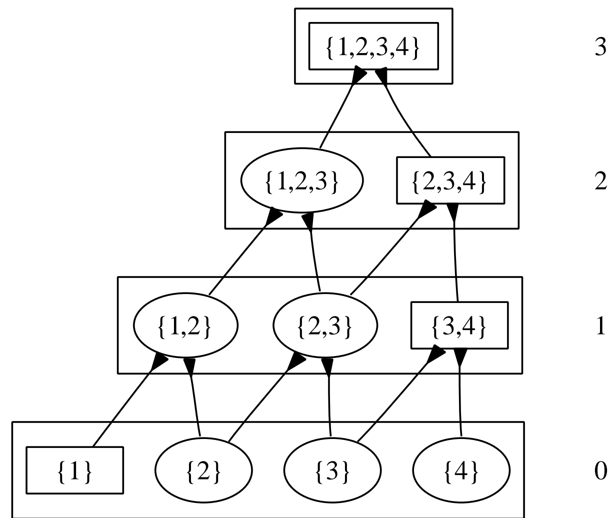


Figure 7.7: Tiling picture of the elevator automaton with 4 states.

3. For each recorded permutation we have to decide exactly which component it belongs to, if there are parallel components on a level. This is done by the the successive approximation (see Section 5.5.1) of the upper fragments. The equivalence class of the resulting set indicates the right component.

After drawing the lookup table, the sets of permutations for each component will be the corresponding generator sets. If there is no entry or there are only constant map entries, then that component's holonomy group is trivial.

7.3.2 Example

An Aperiodic Example: the Elevator Again

Recall Example 4.2. Now we are interested in the dependency functions of the lifts of the generator transformations. To keep it simple we consider the elevator with 3 levels. The generators are $d = (1\ 1\ 2\ 3)$ and $u = (2\ 3\ 4\ 4)$. The skeleton can be seen on Fig. 7.7. The algorithm gives the following lookup tables (using the notation of lookup tables in Example 2.2):

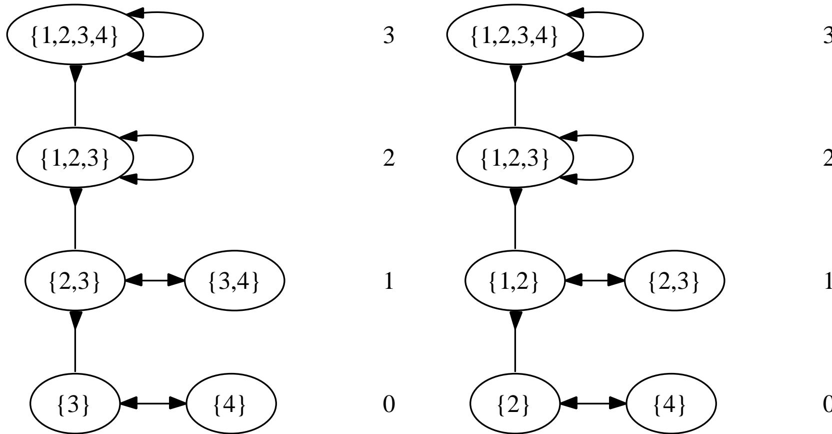


Figure 7.8: State lifts of states 3 (left) and 2 (right) in the holonomy decomposition of the elevator automaton with 4 levels. The bidirectional arrows denote the coding and encoding of tile chains.

$\widehat{u}$:

$$\begin{aligned} f_3^u() &= \text{constant}\{2,3,4\} \\ f_2^u(\{2,3,4\}) &= \text{constant}\{3,4\} \\ f_1^u(\{3,4\},\{2,3,4\}) &= \text{constant}\{4\} \end{aligned}$$

$\widehat{d}$:

$$\begin{aligned} f_3^d() &= \text{constant}\{1,2,3\} \\ f_2^d(\{1,2,3\}) &= \text{constant}\{2,3\} \\ f_1^d(\{2,3\},\{1,2,3\}) &= \text{constant}\{3\} \end{aligned}$$

Note that the indices of the functions correspond to height numbers, f_i gives an action on the ith level.

It is somewhat surprising that there are only few entries in the lookup table: only 3 entries per transformations, while there are 8 (2^3 tile chains) states in the cascaded product. But still, the mappings within the equivalence classes and the successive approximation do the job. Let's consider the following cascaded state: $\hat{a} = (\{4\},\{3,4\},\{1,2,3\})$, which is a lift of 3. Let's apply $\widehat{d}$ to this cascaded state. We know that it should produce a state lift for 2, otherwise the division would not hold. For the the top level (third) we have a constant map defined, so we get $\{1,2,3\}$. On the next (second) level $\widehat{d}$ is defined for the fragment (, ,{1,2,3}), so we apply the constant map yielding $\{2,3\}$. On the bottom level (first) we have no entry in the dependency function lookup table, therefore we leave the existing

component there. We got $\hat{a} \cdot \widehat{d} = (\{4\}, \{2,3\}, \{1,2,3\})$, and that is a lift for 2, as expected (check Fig. 7.8).

Dependency Functions for a Nontrivial Holonomy Group

Recall Eilenberg's example again (Fig. 5.3). We have a nontrivial group for the tiles $B_{\{3,4,5\}}$. Actually it is S_3. Here are the corresponding dependency function entries:

$\widehat{x}$:

$$f_1^x(\{1,2,3\},\{1,2,3,4\}) = (3\ 4\ 5\ 3\ 4\ 4)$$

$\widehat{y}$:

$$f_1^y(\{1,2,3\},\{1,2,3,4\}) = (3\ 5\ 4\ 3\ 5\ 3)$$

The tiles are singleton sets $\{3\}$, $\{4\}$ and $\{5\}$, therefore it is easy to check that one function value is a cyclic permutation, the other one is a transposition. Together they form a minimal generator set for the symmetric group.

In general, the minimality of the generator set is not guaranteed, since we may have to different generators for the original semigroup which act the same way on a subset of the state set.

Narrowing the Skeleton

Observing that since we have a bijection between tile chains and cascaded states and we know the mapping between them (see Section 5.4.2) leads to the idea that we do not need the full skeleton in order to construct the holonomy components. We need only the tilings of the representatives, and tile chains can be recovered from them, if needed. How much percent is this part compared to the size of the full skeleton? Here is a table about the ratio in the case of full transformation semigroups (also see Fig. 7.9).

Semigroup	Ratio
S_2	100%
S_3	85%
S_4	80%
S_5	58%
S_6	39%

This seems to be a very good news, since as $\mathcal{I}$ gets bigger the ratio of the needed subsets versus the size of $\mathcal{I}$ decreases. The bad news is that we cannot exploit this property unless we solve the problem of generating tiles locally (see the problem of nonimage tiles in Section 5.7.3).

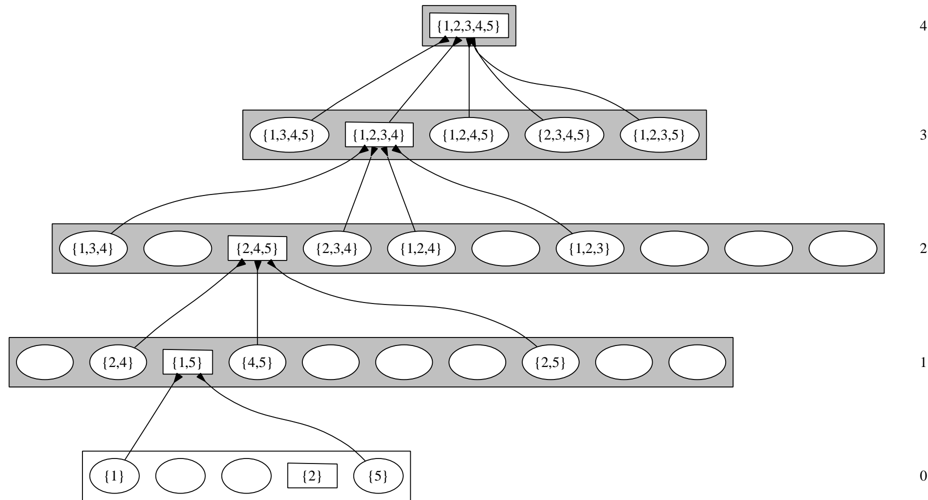


Figure 7.9: The partial tiling picture of $\mathcal{T}_5$. Only the equivalence class representatives and their tiles are displayed. These are needed for carrying out transformations in the wreath product semigroup. The other elements of $\mathcal{I}$ can be generated by mapping cascaded states to tile chains.

7.4 Summary

Constructing the holonomy component is a critical issue in implementing the holonomy decomposition. Here we described two different methods. The first one is a more advanced search than the brute-force approach. It uses the knowledge about the structure of tiling and its relations to the equivalence classes. Exploring these connections improved our understanding of holonomy decomposition. However, regarding its efficiency in terms of a small generator set, it is still not satisfying. Therefore we presented another approach in a completely different manner. Leveraging the dependency functions we get a generator set comparable in size to the original automaton's generator set.

Chapter 8

Applications

"The engine turns, the Maker rests."
Imre Madách, *The Tragedy of Man*, 1860.

The hierarchical decomposition of automata gives us wonderful promises regarding its applicability: automated object-oriented programming in software development [Neh94], formal methods for understanding in artificial intelligence [Neh96a]. The least number of needed levels for decomposition provides a widely applicable integer-valued complexity measure, including applications ranging from electrical engineering and physics to evolutionary biology [Rho71, NR00, Neh96a, Arb68], just to briefly mention some important keywords. We also have now working implementations, therefore it is reasonable to ask what happens to the promises. Why are the foretold revolutionary results not delivered yet? There are three basic reasons:

1. Since this work gave the first computational implementation, simply due to the lack of tools before no one has tried to do practical applications before.

2. Our knowledge of the detailed inner workings of actual decompositions is still rudimentary.

3. The implementations are not yet scalable.

Our assumption is that the second reason is due to the first one. Therefore we think that by studying small but nontrivial examples, understanding otherwise well-known structures differently, via hierarchical decomposition, eventually will lead us to scalable implementations by exploiting the special properties of the decompositions.

Here we take this route, first we will show some features of the holonomy decomposition emphasizing the role of the subduction equivalence classes using spatial clues. Then we decompose finite residue class rings modulo n represented as semigroups. And finally we discuss the difficulties of applying hierarchical decompositions as formal models of understanding.

8.1 Understanding the Holonomy Decomposition

The holonomy decomposition gives us a hierarchically structured finite automaton, but it is not immediate how this cascaded automaton works, and how it is related to the original automaton although a division is constructed in the proof of Theorem 5.1. Therefore it is worth examining illustrative examples in order to understand how in detail the holonomy decomposition provides hierarchical coordinate systems on finite automata. Since we are human beings living in a three-dimensional physical world, it is easier for us to comprehend abstract constructs in spatial terms. Here we will first consider state transition diagrams with some specific forms. The nodes represent points in space and the edges represent the connections in that space, thus input symbols correspond to movements in the space.

8.1.1 Decompositions Without any Hierarchical Dependence

The most basic examples are with only one hierarchical level. These are the automata whose characteristic semigroups are groups (possibly with constants). As all input symbols are permutations or resets $\mathcal{I}$ consists of the full state set and the singletons.

Torus. The torus is basically a $m \times n$ grid of points with a wrap-around (Fig. 8.1.1). Since we can get back to the a state by going into one direction, we have loops, thus we have nontrivial groups. Wherever we are, we can reach any other point, thus there is no irreversibility of the operations and this yields the very simple holonomy decomposition: the only component of the decomposition is $(\mathbf{mn}, \overline{C_m \times C_n})$.

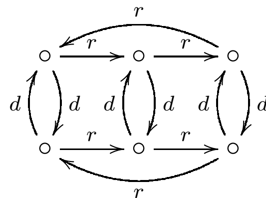


Figure 8.1: A 2×3 toroidal grid as a state transition graph of an automaton. States are denoted by the circle nodes (without names, but can be addressed by coordinates). The input symbols are d and r, corresponding to moves on the grid: down and right.

8.1.2 The Role of Subduction Equivalence Classes

The role of equivalence subduction classes is that we do not have to consider explicitly those elements of $\mathcal{I}$ which behave the same way under the action of

S, i.e. their holonomy groups are isomorphic and there are transformations in S to establish this isomorphism.

Irreversible Mesh. Let's consider automata with transition graphs of shape of a bounded rectangular plane, and with two input symbols going left and down. Clearly, these automata are aperiodic, since applying any of the input symbols collapses states towards the left or bottom edges. Moreover, each element of $\mathcal{I}$ forms a subduction equivalence class on its own. The

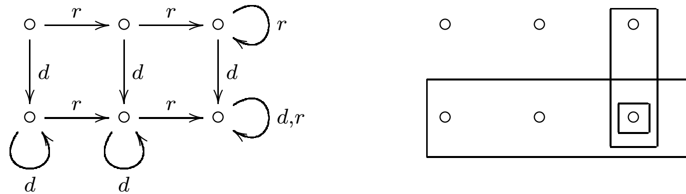


Figure 8.2: A 2×3 irreversible mesh automaton. On the left are 3 example elements of the set of the images. The images are rectangles with the common bottom right corner. Each of them form a singleton equivalence class on its own, due to the fact that we cannot move backwards in either direction.

example on Fig 8.2 shows a 2×3 mesh with input symbols r, d for moving right and down on the grid. The bottom-right corner is the state where eventually the movement ends up, therefore this state is contained in all elements of $\mathcal{I}$ (except the other singletons), which are basically the rectangles with the common bottom-right corner. Its decomposition is $(\overline{1_2 \times 1_2}) \wr (\overline{1_2 \times 1_3}) \wr \overline{1_3}$, $|\mathcal{I}| = 11$.

Partially Reversible Mesh. The situation changes when we have the other directions as well (going up and right) as cycles appear. Despite the cycles in the graph it is still aperiodic (for a further discussion determining aperiodicity in automata see Section 7.2.3). But the equivalence classes are not singletons any more. They consist of all rectangles of the same dimensions.

The example of 2×3 mesh on Fig. 8.3 shows the automaton and one particular equivalence class. Its decomposition is $(\overline{1_2 \times 1_2}) \wr (\overline{1_2 \times 1_4}) \wr \overline{1_4}$, $|\mathcal{I}| = 18$. Although there are more elements in $\mathcal{I}$ than in the previous irreversible case, the decomposition is quite similar, since the equivalence classes are not singletons.

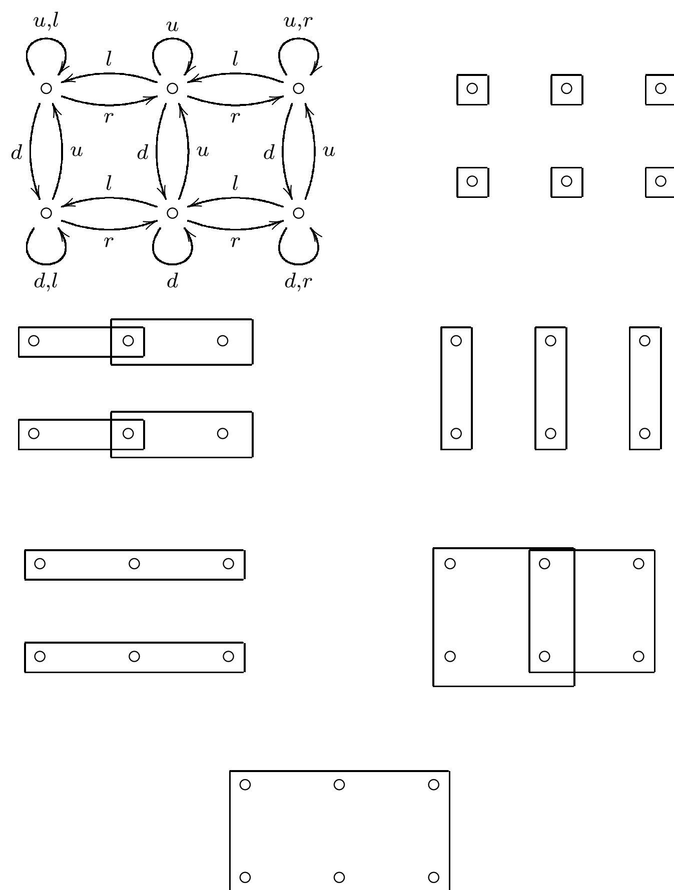


Figure 8.3: A 2×3 partially reversible automaton with all its equivalence classes. An equivalence class has elements that are rectangles with the same dimension.

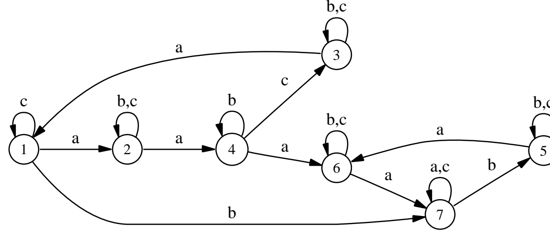


Figure 8.4: Counterexample automaton with 7 states yielding long holonomy decomposition (20 levels). The generators are: $\{(2\ 4\ 1\ 6\ 6\ 7\ 7),(7\ 2\ 3\ 4\ 5\ 6\ 5),(1\ 2\ 3\ 3\ 5\ 6\ 7)\}$. Its decomposition is $(\mathbf{2},\overline{S_2}) \wr (\mathbf{2},\overline{1_2}) \wr (\overline{(\mathbf{2},1_2)\times(\mathbf{3},1_3)}) \wr (\mathbf{3},\overline{1_3}) \wr (\overline{(\mathbf{3},1_3)\times(\mathbf{3},1_3)}) \wr (\overline{(\mathbf{3},1_3)\times(\mathbf{3},1_3)\times(\mathbf{3},1_3)}) \wr (\overline{(\mathbf{3},1_3)\times(\mathbf{4},1_4)}) \wr (\overline{(\mathbf{2},1_2)\times(\mathbf{3},1_3)\times(\mathbf{4},1_4)}) \wr (\mathbf{2},\overline{1_2}) \wr (\mathbf{2},\overline{1_2}) \wr (\mathbf{4},\overline{1_4}) \wr (\mathbf{4},\overline{1_4}) \wr (\overline{(\mathbf{3},1_3)\times(\mathbf{4},1_4)}) \wr (\overline{(\mathbf{3},1_3)\times(\mathbf{3},1_3)\times(\mathbf{4},1_4)}) \wr (\overline{(\mathbf{2},1_2)\times(\mathbf{3},1_3)\times(\mathbf{4},1_4)}) \wr (\overline{(\mathbf{3},C_3)\times(\mathbf{4},1_4)}) \wr (\overline{(\mathbf{3},1_3)\times(\mathbf{3},1_3)}) \wr (\mathbf{3},\overline{1_3}) \wr (\mathbf{2},\overline{1_2}) \wr (\mathbf{3},\overline{1_3})$.

8.2 Properties of the Holonomy Decomposition

8.2.1 Number of Hierarchical Levels

For finite semigroups the number of hierarchical levels is clearly bounded, but it is an important question how does it depend on the number of states of the original automaton. *What is the maximal number of hierarchical levels of the holonomy decomposition of an automaton with n states?* The answer gives the number of coordinates which is the size of the generated formal model.

We have seen already that using the $V \cup T$ method gives us extremely long decompositions, but the holonomy method seems to be a better candidate. By the holonomy method we mean the constructive proof described in this work in Chapter 5, since Zeiger's original method [Zei67] differs in length (generally gives longer decompositions), as the parallel components appear on different levels in his proof.

The first guess would be probably at most $n-1$, according to the intuition that height numbers correspond to cardinalities. This turns out to be false, due the fact that strict subduction is possible between sets with the same cardinalities (see Section 5.7.4).

Since the theoretical approach seems difficult, we can get some help from the computational tool: by using a method resembling genetic algorithms [Hol75] we could find counterexamples. We started from a randomly generated automaton, explored the one point mutation neighbors of this automaton by randomly changing the image of a single element in a generator transformation (i.e. the target of a single arrow in the state-transition dia-

gram), then selected for the longest decomposition and repeated the same cycle. We found experimentally the following state set size and height number pairs: $(5,11),(6,15)$, $(7,20)$ – see Fig. 8.4, and $(8,26)$. Hence even $3n+1$ does not bound $h(\mathcal{A})$. Note that the holonomy decomposition is still more efficient than the $V \cup T$ technique in terms of the length of the wreath product (see [ENN04] or Chapter 4).

We do not know an exact bound for the length of the holonomy decompositions yet, but we can summarize the observations of our genetic algorithm experiments.

Observation 8.1 *Long holonomy decompositions tend to have a low number of nontrivial holonomy group components with small cardinality.*

It seems that in order to build a high skeleton, we need sufficiently many elements in $\mathcal{I}$, and that is provided by the nontrivial group components' permutations. But on the other hand, if we have a group component with high order, then its subgroups might also be components on lower levels, thus collapsing the hierarchy.

8.2.2 Size of a Component's State Set

Another important question is *How many a tiles can a set Q in $\mathcal{I}$ have given its cardinality?* The answer gives the number of states of a component, contributing to the size of the state set of the cascaded product. Intuition might say that if we have more tiles, then they should become inclusion related, thus not satisfying the definition of being a tile, so that a subset $Q \in \mathcal{I}$ should not have more tiles than its cardinality as for the full ts.

However, mistyping one character the generators of R_6 (the semigroup of integers modulo 6, see Section 8.3) yielded an automated decomposition, where the top level set with cardinality 6 has 14 tiles (see Fig. 8.5). This example also shows the ruggedness of the landscape of the decompositions, since just changing one single image of a transformation results in a completely different hierarchical structure and sets of tiles.

The analysis of the example in Fig. 8.5 shows that the high number of tiles for the top level component is due to the missing sets with cardinalities between the top and next level below. This suggests set theoretical considerations: What is the largest number of tiles that are not subsets of each other? Considerations based on examples led to the following conjecture:

Conjecture 8.2 $\forall Q \in \mathcal{I},\quad |B_Q| \leq \binom{n}{\lfloor \frac{n}{2} \rfloor}$ *where* $n = |Q|$.

Actually we can show that it is indeed possible that $|B_Q| = \binom{n}{\lfloor \frac{n}{2} \rfloor}$. We need the generators of the symmetric group S_n:

$$(2\ 3\ \ldots\ n-1\ 1), (2\ 1\ 3\ \ldots\ n)$$

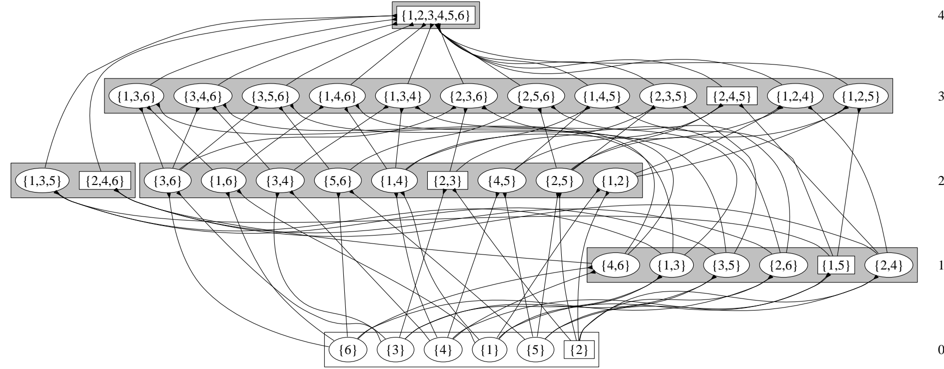


Figure 8.5: The tiling picture of a counterexample automaton with many tiles at the top level in decomposition. Generators: $\{(2\ 3\ 4\ 5\ 6\ 1), (1\ 3\ 5\ 1\ 3\ 5), (1\ 4\ 1\ 4\ \mathit{3}\ 4), (1\ 6\ 5\ 4\ 3\ 2)\}$. Its decomposition is $(\mathbf{2}, \overline{S_2}) \wr (\overline{(\mathbf{2}, S_2) \times (\mathbf{3}, S_3)}) \wr (\mathbf{2}, \overline{S_2}) \wr (\mathbf{14}, \overline{D_{12}})$. The description of this ts differs from R_6 only in the image of a single element under a single generator (italicized).

and an arbitrary transformation t which collapses $\lceil \frac{n}{2} \rceil$ states, thus its rank is $\lfloor \frac{n}{2} \rfloor$. For instance a transformation t given by:

$$t(i) = \begin{cases} 1 & t \leq \lceil \frac{n}{2} \rceil \\ i & \text{otherwise} \end{cases}$$

In the ts on $\mathbf{n}$ generated by these three transformations the tiles of $\mathbf{n}$ are all subsets with $\lfloor \frac{n}{2} \rfloor$ elements.

The fact that subduction as a partial order contains the inclusion relation suggests that reasoning about the inclusion partial order of $\mathcal{I}$ gives us an upper bound.

Proposition 8.3 *Let B_Q be the set of tiles Q in a holonomy decomposition and $|Q| = n$, then $|B_Q| \leq \binom{n}{\lfloor \frac{n}{2} \rfloor}$.*

Proof: Since $(2^Q, \subseteq)$ has a maximal antichain consisting of all subsets with $\lfloor \frac{n}{2} \rfloor$ elements, Dilworth's Theorem (see Appendix B) implies that the number of chains needed to cover $(2^Q, \subseteq)$ equals to $\binom{n}{\lfloor \frac{n}{2} \rfloor}$. Since $\mathcal{I}$ does not necessarily equals 2^Q (it is a subset of it), we need the same number of or less chains to cover $(\mathcal{I}\downarrow_Q, \subseteq)$, the elements of $\mathcal{I}$ below Q in the inclusion relation, i.e. the subsets of Q. The number of chains covering $(\mathcal{I}\downarrow_Q, \subseteq)$ is at least the number of the maximal subsets of Q, which are the tiles of Q by definition. □

Since we know that this maximal number of states for a component is achievable, we have a sharp bound.

8.3 Decomposition of the Rings of Integers Modulo n

The transformation semigroup of finite residue class ring of integer modulo n is a set $\mathbf{n}$ with the operations of addition and multiplication modulo n, and is denoted by R_n.

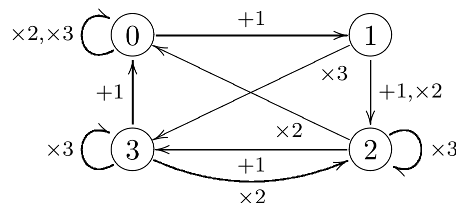
8.3.1 Representation

Integer modulo n residue class rings R_n can be represented by automata. The state set is $\mathbf{n}$, the residue classes modulo n. The transformations correspond to the operations of the ring represented as one-argument functions such as $+1, \times 2, \times 3$ and so on. As generators, clearly we need only at most $+1$ and the multiplications by the prime elements smaller than n. The characteristic semigroup of this automaton is denoted by $S(R_n)$.

Proposition 8.4 $|S(R_n)| = n^2$.

Proof: $S(R_n)$ is a noncommutative semigroup, a semidirect product of the additive group C_n and the multiplicative monoid in R_n, since the elements are given by the affine linear transformations of the form $\times k + l$ which are closed under composition. It is easy see that all distinct pairs $k, l \in \mathbf{n}$ give distinct transformations of $\mathbf{n}$, therefore we have n^2 transformations. □

For example, the following automaton represents the R_4, the residue class ring of integers modulo 4:



Since it is easy to move between the ring and semigroup notation, we will use the notation $\alpha m + \beta$ instead of $(m \cdot \times\alpha) \cdot +\beta$, where $m \in \mathbf{n}$ and α, β are elements of the ring R_n.

8.3.2 The Extended Set of Images

The extended image set $\mathcal{I}$ consists of the sets given in the form $\alpha\mathbf{n} + \beta$. The additive factor always induces a permutation on $\mathcal{I}$ and the multiplicative factor will also induce a permutation if and only if (α, n) are relative primes. If α shares a divisor with n then collapsing of states occurs. Therefore the set $\mathcal{I}$ of proper images of $S(R_n)$ consists of the subsets $\alpha\mathbf{n} + \beta$ of $\mathbf{n}$, where $\gcd(\alpha, n) \neq 1$ and $0 \leq \beta < \alpha$. To make these intuitive statements more exact, we have the following facts:

Fact 8.5 *Suppose $\alpha \mid n$, $\alpha \in \boldsymbol{n}$, then multiplication by a prime p collapses some elements of $\alpha\boldsymbol{n}$ if $p\alpha \mid n$, but permutes the elements of $\alpha\boldsymbol{n}$ if $p\alpha \nmid n$.*

Proof: If $p\alpha \mid n$, then $|\alpha\mathbf{n}| = \frac{n}{\alpha}$ and $|p\alpha\mathbf{n}| = \frac{n}{p\alpha}$, thus $\langle p\alpha\rangle \subsetneq \langle\alpha\rangle$, where $\langle\alpha\rangle$ is the principal ideal generated by α. Due to the strict inclusion we have collapsing by p.

$p\alpha \nmid n$ case: Let $k\alpha, l\alpha \in \langle\alpha\rangle$ and assume that $0 \leq, k\alpha, l\alpha < n$. Now suppose this is true $pk\alpha = pl\alpha$. Then $p\alpha(k-l) = qn$, and $p \mid q$ since $\alpha \mid n$ and $p\alpha \nmid n$. So writing $q = q'p$, we have $pk\alpha - pl\alpha = pq'n$ and dividing by p we get $k\alpha - l\alpha = q'n$, thus $k\alpha \equiv l\alpha \mod n$. According to the assumptions $k\alpha$ and $l\alpha$ are smaller than n, therefore $k\alpha = l\alpha$, thus $\times p$ permutes $\alpha\mathbf{n}$. □

Next, we show that other integers that are not divisors of n do not produce new images.

Fact 8.6 *For any integer β, $\beta\boldsymbol{n} \equiv \alpha\boldsymbol{n}$ for some $\alpha \mid n$. In fact $\alpha = \gcd(\beta, n)$.*

Proof: Let $\beta = \alpha k$, $k = p_1 \cdots p_m$ (with repetitions possible; and the case $k = 0$ is obvious). If we choose α as the greatest common divisor of β and n, then $p_i\alpha \nmid n$. Applying Fact 8.5 to each p_i m times yields that k permutes $\alpha\mathbf{n}$, thus $\alpha\mathbf{n} = k\alpha\mathbf{n} = \beta\mathbf{n}$. □

8.3.3 Subduction, Equivalence Relation, and the Tiling Picture

Here we show how the subduction and the equivalence relations are connected to the operations in R_n. The equivalence classes are determined by the multiplicative factor and the elements of a class are determined by the additive factor. Moreover, the partial order of the equivalence classes is the same as the inclusion relation of the principal ideals. Here we use the fact that $\langle\alpha\rangle = \alpha\mathbf{n}$, where $\langle\alpha\rangle$ is the principal ideal of the ring generated by α. The equivalence relation is easier to grasp since it is related to the additive operations.

Lemma 8.7 $\alpha\boldsymbol{n} \equiv \alpha\boldsymbol{n} + \beta$.

Proof: To $\alpha\mathbf{n}$ we can apply the transformation $+\beta$, and to $\alpha\mathbf{n} + \beta$ we can apply the transformation $-\beta$, thus establishing the equivalence. □

Now we can proceed to understand how the multiplicative operations are connected to the subduction relation.

Lemma 8.8 *Let $P = \alpha_0\boldsymbol{n} + \beta_0$, $Q = \alpha_1\boldsymbol{n} + \beta_1$ be elements of $\mathcal{I}$, then $P \leq Q \Longleftrightarrow \alpha_1 \mid \alpha_0 \pmod{n}$.*

Proof: According to the definition of subduction relation $P \le Q$ if $P \subseteq Q \cdot s$ for some $s \in S(R_n)$, i.e. $P \subseteq \alpha Q + \beta$. Thus we have $\alpha_0\mathbf{n} + \beta_0 \subseteq \alpha(\alpha_1\mathbf{n} + \beta_1) + \beta$, or equivalently $\alpha_0\mathbf{n} \subseteq \alpha\alpha_1\mathbf{n} + \beta_2$. We know that 0 and α_0 are in $\alpha_0\mathbf{n}$, since $0, 1 \in \mathbf{n}$.

$$\begin{aligned} 0 &= \alpha\alpha_1 i + \beta_2 \bmod n \\ \alpha_0 &= \alpha\alpha_1 j + \beta_2 \bmod n \end{aligned}$$

for some $i, j \in \mathbf{n}$. Subtracting the first equation from the second we get $\alpha_0 = \alpha\alpha_1(j - i) + kn$ for some k. Therefore $\alpha_1 \mid \alpha_0 \bmod n$.

If $\alpha_1 \mid \alpha_0$ then $\alpha_0 = \alpha\alpha_1 + kn$ for some k, thus $\alpha_0\mathbf{n} = (\alpha\alpha_1 + kn)\mathbf{n} = \alpha\alpha_1\mathbf{n}$. Then $P = \alpha_0\mathbf{n} + \beta_0 = \alpha\alpha_1\mathbf{n} + \beta_0 = \alpha(\alpha_1\mathbf{n} + \beta_1) + (\beta_0 - \alpha\beta_1) = \alpha Q + \beta'$. □

Note that in the second part of the proof we have equality, not just inclusion.

We still have to show that there are no elements equivalent to $\alpha\mathbf{n}$ except those of the form $\alpha\mathbf{n} + \beta$, i.e. the equivalence classes correspond exactly to the principal ideals of the ring.

Lemma 8.9 *Let $Q = \alpha\boldsymbol{n} + \beta$, $Q' = \alpha'\boldsymbol{n} + \beta'$ and $Q' \equiv Q$, then $\alpha\boldsymbol{n} = \alpha'\boldsymbol{n}$.*

By Lemma 8.7 we have $\alpha' \mid \alpha$ and $\alpha' \mid \alpha \pmod n$, thus $\alpha = \zeta\alpha' \in \langle\alpha'\rangle$ and $\alpha' = \chi\alpha \in \langle\alpha\rangle$ for some $\zeta, \chi \in R_n$, therefore $\langle\alpha'\rangle \subseteq \langle\alpha\rangle$ and $\langle\alpha\rangle \subseteq \langle\alpha'\rangle$ yielding $\langle\alpha\rangle = \langle\alpha'\rangle$. Using this result we get $Q' = \alpha'\mathbf{n} + \beta' = \langle\alpha'\rangle + \beta' = \langle\alpha\rangle + \beta' = \alpha\mathbf{n} + \beta' \equiv Q$. □

We can summarize the previous results in the following theorem about the tiling picture of $S(R_n)$. As the choice of the representative is arbitrary, we may take it to have zero as the additive factor, thus to have the *canonical form* $\alpha\mathbf{n}$.

Theorem 8.10 *Let $n = p_1^{\nu_1} \cdots p_m^{\nu_m}$, and P, Q be elements of $\mathcal{I}$ with representatives $\overline{P} = \alpha\boldsymbol{n}$ and $\overline{Q} = \beta\boldsymbol{n}$, where $\alpha = p_1^{\alpha_1} \cdots p_m^{\alpha_m}$ and $\beta = p_1^{\beta_1} \cdots p_m^{\beta_m}$, $0 \le \alpha_i, \beta_i \le \nu_i$, then*

1. *$P \le Q$ if and only if $\alpha_i \ge \beta_i$ for all i with $1 \le i \le m$,*
2. *$P < Q$ if and only if $\alpha_i \ge \beta_i$ for all i and $\alpha_i > \beta_i$ for some i,*
3. *$P \equiv Q$ if and only if $\alpha = \beta$.*

Proof: The statements of the theorem follow from Lemmas 8.7, 8.8, and 8.9. □

In the notation of Theorem 8.10,

Corollary 8.11 *$\overline{P}$ is a tile of $\overline{Q}$ if and only if $\alpha_i = \beta_i + 1$ for some i and $\alpha_j = \beta_j$ for all $j \neq i$.*

In the vein of Facts 8.5 and 8.6, we can reformulate the description of the tile of relation:

Fact 8.12 *Suppose $\alpha \mid n$ and $\beta \mid n$, then*

$$\beta\boldsymbol{n} \prec \alpha\boldsymbol{n} \Longleftrightarrow \beta = p\alpha \text{ for some prime } p.$$

8.3.4 Number of Levels

In studying hierarchical decompositions one of the key questions is the number of hierarchical levels in the cascaded product, or more precisely in the case of the holonomy decomposition the height of the automaton.

Theorem 8.13 *The height h of the decomposition of R_n is*

$$h = \sum_{i=1}^{k} \nu_i$$

where $n = p_1^{\nu_1} \cdots p_m^{\nu_m}$ is the prime factorization for n.

Proof: We showed that the strict subduction and the inclusion relations are along the multiplications by prime factors, therefore the maximum length of strict chains in the tiling picture is the maximum length of the products of the prime factors with multiplicity. □

8.3.5 Number of States

Another important question is how many points the holonomy components act on. In the case of R_n observation suggested the following theorem:

Theorem 8.14 *Let $(\mathcal{B}_h, \overline{\mathcal{H}_h})$ be the top level component of the decomposition of R_n with height h, then*

$$|\mathcal{B}| = \sum_{i=1}^{k} p_i$$

where $n = p_1^{\alpha_1} \cdots p_k^{\alpha_k}$ is the prime factorization for n.

Proof: The tiles of $\mathbf{n}$ are of the form $p\mathbf{n}+k$, where p is a prime divisor of n and $k \in \mathbf{p}$. If p is not a prime, then the maximality condition of a tile is not satisfied. Also, if p is prime but not a divisor of n, then $p\mathbf{n} = \mathbf{n}$, so $p\mathbf{n}$ is not a tile. Since the elements of an equivalence class are determined by addition (Lemma 8.7 and 8.9), the class consists of cosets of $p\mathbf{n}$ that are $p\mathbf{n}+k$, $k \in \mathbf{p}$, since any element can be reached from any other just by using addition modulo p. Therefore, an equivalence class of p has exactly p elements. The equivalence classes induced by different primes cannot be subduction related, since they are relative primes (see also Lemma 8.8). Therefore we have as many equivalence classes as many prime divisors, and each equivalence has as many elements as its corresponding prime, thus we have the sum as in the theorem. □

This can be generalized by considering the fact that the equivalence class representatives (the canonical ones in the form of $\alpha\mathbf{n}$ with $\alpha \mid n$) correspond to principal ideals.

Theorem 8.15 *Let $Q = \alpha\boldsymbol{n}$, $Q \in \mathcal{I}$ a canonical equivalence class representative, and its holonomy component be (B_Q, H_Q). Then the number of states is given by*

$$|B_Q| = \sum p_i$$

where $\frac{n}{\alpha} = p_1^{\alpha_1} \cdots p_k^{\alpha_k}$ is the prime factorization for $\frac{n}{\alpha}$.

Proof: The key point of the proof is to show where does $\frac{n}{\alpha}$ come from. Using Fact 8.12, $p_i\alpha\mathbf{n} \prec \alpha\mathbf{n}$ iff $p_i\alpha \mid n$, then dividing by α we get $p_i \mid \frac{n}{\alpha}$. Then as in Theorem 8.14 we count the number of tiles. □

8.3.6 Holonomy Group Components

Now we can characterize the invertible elements of $S(R_n)$.

Proposition 8.16 *$\pi \in S(R_n)$ is invertible iff π can be represented as $ax + b$, $0 < a < n$, $\gcd(a, n) = 1$.*

We also can give a small generator set for each holonomy group component.

Proposition 8.17 *Let $Q = \alpha\boldsymbol{n}$, $Q \in \mathcal{I}$ a canonical equivalence class representative, and its holonomy component be (B_Q, H_Q). Then H_Q is generated by the set of transformations defined by:*

$$\{ax : \gcd(a, \frac{n}{\alpha}) = 1\} \cup \{+1\}.$$

Now we present a theorem that basically says that the decomposition of $S(R_n)$ can be built from the unique top level components of the decompositions of certain smaller ones.

Theorem 8.18 *Let $\alpha \mid n$ and $\alpha\boldsymbol{n}$ be a canonical representative of an equivalence class in the tiling picture of $S(R_n)$. Then the holonomy group component of $\alpha\boldsymbol{n}$ is isomorphic to the holonomy group component of $\boldsymbol{m}$ in the decomposition of $S(R_m)$,i.e.*

$$(B_{\alpha\boldsymbol{n}}, H_{\alpha\boldsymbol{n}}) \cong (B_{\boldsymbol{m}}, H_{\boldsymbol{m}})$$

where $m = \frac{n}{\alpha}$.

Proof: We show the isomorphism, that explicit mappings from the tiles and the holonomy group of $\mathbf{m}$ form a homomorphism which is one-to-one and onto. In this case it will suffice to consider the action of the holonomy group elements on $\mathbf{m}$ and $\alpha\mathbf{n}$.

Bijections. The two state sets are the following

$$\mathbf{m} = \{0, 1, \dots, \frac{n}{\alpha} - 1\}$$

and

$$\alpha\mathbf{n} = \{0, \alpha, \dots, n - \alpha\}$$

Let's consider the function $\phi(i) := \alpha i$ which is obviously bijective. ϕ induces a bijection of the powersets of $\alpha\mathbf{n}$ and $\mathbf{m}$ as well. It follows that it also induces a bijection of tiles as well (for details see Lemma 5.7).

Let $\pi \in H_{\mathbf{m}}$, $\pi : x \mapsto ax + b \bmod m$, where for all $x \in \mathbf{m}$ $0 \leq a, b \leq m - 1$, $a \neq 0$, and $\gcd(a, m) = 1$ for all $x \in \mathbf{m}$, otherwise it collapses according to Fact 8.5. Define $\phi : \pi \mapsto \pi^*$ as such that $\pi^* : y \mapsto ay + \alpha b \bmod n$ for all $y \in \alpha\mathbf{n}$. π^* is also bijective on $\alpha\mathbf{n}$.

ϕ is a homomorphism. We may assume:

Cond1 : $0 \leq b, d < \frac{n}{\alpha}$

Cond2 : $0 < a, c < \frac{n}{\alpha}$

Cond3 : $\gcd(a, \frac{n}{\alpha}) = 1$, $\gcd(c, \frac{n}{\alpha}) = 1$

Let $\pi = ax + b, \rho = cx + d \in H_{\mathbf{m}}$. We need to show that $\pi^*\rho^* = (\pi\rho)^*$.

$$\begin{aligned}((i\phi)\pi^*)\rho^* &= c(a(\alpha i) + \alpha b) + \alpha d \\ &= ca\alpha i + \alpha cb + \alpha d\end{aligned}$$

Note that $i\phi$ is an arbitrary element of $\alpha\mathbf{n}$.

$$\begin{aligned}(i\phi)(\pi\rho)^* &= (i\alpha)(c(ax + b) + d)^* \\ &= (i\alpha)(cax + bc + d)* \\ &= ca\alpha i + \alpha cb + \alpha d\end{aligned}$$

ϕ is one to one. Suppose $\pi^* = \rho^*$, i.e. $ay + \alpha b = cy + \alpha d \bmod n$, $y \in \alpha\mathbf{n}$.

Taking $y = 0$:

$$\alpha b = \alpha d \text{ mod } n,$$

we have

$$0 \leq \alpha b, \alpha d < n$$

due to Cond1. Since they are equivalent mod n and smaller than n, therefore $b = d$.

So replacing d by b we have

$$\begin{aligned} ay + \alpha b &= cy + \alpha b \\ ay &= cy \end{aligned}$$

Taking $y = \alpha$:

$$a\alpha = c\alpha \text{ mod } n$$

Again (due to Cond2 multiplied by α), $a = c$. Thus $\pi = \rho$.
ϕ **is onto.** Let $\gamma = ay + b$, $\gamma \in H_{\alpha\mathbf{n}}$. We show that γ is an image of some π: $\gamma = \pi^*$. It is immediate that $\alpha|b$, by considering the case of $y = 0$. Take $\pi = ax + \frac{b}{\alpha}$. Suppose π collapses states: $ai = aj$ with $i \neq j,\ i, j \in \mathbf{m}$, then $a\alpha i = a\alpha j$ mod n, so $\alpha i \neq \alpha j$ mod n, whence $\alpha i \neq \alpha j$. Contradiction!
This establishes the isomorphism of permutation groups. □

8.3.7 The Lesson

The reason why we chose these rings for studying their decompositions is that they are quite regular (hence the nice structure theorem), but they are not trivial as well. We think that the results can be generalized or act as a guiding metaphor.

The distinction between the additive and the multiplicative factor is important in building the skeleton. Roughly speaking, the multiplicative factors determine the poset of equivalence classes, while the additive factors fill the equivalence classes. For future research, the question is whether this distinction can be generalized to the distinction of permutations and collapsing transformations for an arbitrary ts.

8.4 Formal Models of Understanding

The basic idea of applying decompositions as the formal models of understanding is that using the cascaded product we can answer questions about the original automaton in a very convenient way. The main idea is that we can use the wreath product as a coordinate system and the elements of the components as values for the coordinates.

8.4.1 Representations in Artificial Intelligence

According to the so-called Good Old-Fashioned Artificial Intelligence approach we have to build a system with a reasonably accurate representation of its environment to make it behave intelligently. But this just does not work. The hardwired model is rigid, even cannot cope with small changes of the environment, or it the representation should contain all details with all the possible changes, thus combinatorial explosions pop up. Therefore the Artificial Intelligently (AI) community has come up with the strange idea that we do it better without any representation [Bro99]. Clearly, this is a fruitful method showing that one can have complex behavior without complex inner structure. But it is also clear that we cannot get too far without representations [Ste03]. Here we adopt the viewpoint that we often need representations of the environment in order to realize artificial intelligence, but the representation should be flexible and dynamically changing over time and obtained by the artificial system on its own by recognizing regularities of the real world around.

8.4.2 The 'What to do?' Problem

The main problem to be solved here is quite a natural one: given two states of the original automaton, one is the state in which we are currently, the other one is the state we would like to reach. The task is to give a word of input symbols (sequence of actions) that induces the desired transformation of states, if it the target state is reachable at all. One can imagine any kind of intelligence (natural/artificial/alien) facing this kind of problem: knowing what is the situation now and what is the desired goal, the intelligence has to figure out what operations to carry out.

Of course for single states this can be done in the original automaton by finding a path, but for subsets it would be lot more complicated. Also, here we are interested in using the formal model of understanding *instead* of the unanalyzed original automaton.

A good example is the algorithm we apply when we perform adding numbers in decimal notation. Given the first operand and the result we can calculate the second operand. However, this example is not general enough, since decimal notation is a cascaded product of numbers, although in the general case we do not necessarily have inverses all the time.

Buses and Trains

Using the skeleton we can answer some questions regarding the automaton quite quickly. This way we show directly how the underlying structure of the decomposition works. Here we proceed in solving the problem without using the coordinate system.

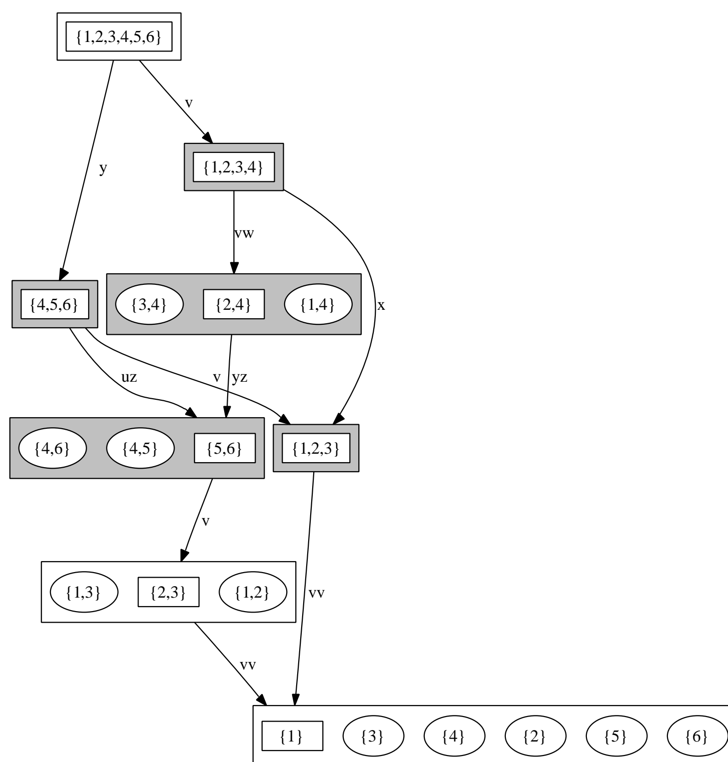


Figure 8.6: The partially ordered set of equivalence classes in the image relation for the decomposition of the automaton discussed in Section 5.7.3. The arrows denote the image relation of the class representatives (the trains, see in the text). A label denotes a witness.

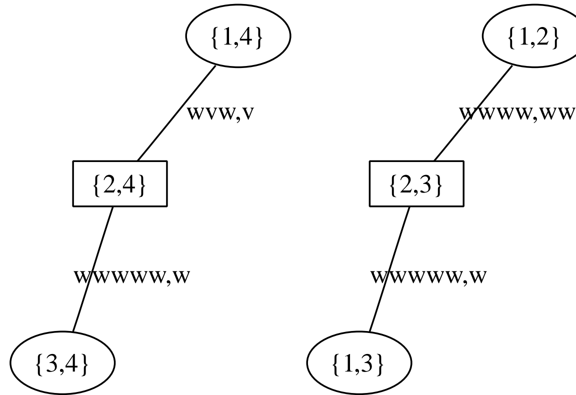


Figure 8.7: The buses for two equivalence classes. The first part of the label denotes the mapping from a member to the representative, the second is the map backwards. For instance, for going from $\{3,4\}$ to $\{2,4\}$ we apply the word $wwwww$, from $\{2,3\}$ to $\{1,2\}$, we apply $wwww$. Recall that $w = (2\ 3\ 1\ 4\ 4\ 4)$ permutes $\{1,2,3,4\}$. Also note, that these words given by the algorithm are not the shortest possible ones.

The leading metaphor is the following. If we would like to travel from one village to another one far away, then first we take the local bus to the closest train station, take the train to the station closest to the destination village and catch a local bus again. Similarly, given $P, Q \in \mathcal{I}$, (the two distant villages), we need to make the following steps.

1. We go from P to $\overline{P}$ by the map $\overrightarrow{m}_P$:

$$P \cdot \overrightarrow{m}_P = \overline{P}.$$

2. If there exists a witness w in the image relation for $\overline{P} \trianglelefteq \overline{Q}$, then we go to the equivalence class representative of Q:

$$\overline{P} \cdot w = \overline{Q}.$$

   If $\overline{Q}$ is not an image of $\overline{P}$, then we cannot go to Q from P.

3. We go from $\overline{Q}$ to Q by $\overleftarrow{m}_Q$:

$$\overline{Q} \cdot \overleftarrow{m}_Q = Q.$$

4. Finally we combine the maps (or concatenate the corresponding words) and get the required transformation

$$P \cdot \overrightarrow{m}_P \cdot w \cdot \overleftarrow{m}_Q.$$

The required mappings are calculated during the decomposition, therefore we get the answer in constant time. In the case of singletons this is the same problem as finding a path in the state transition graph of the automaton.

Let's see this on an example. Consider the tiling picture on Fig. 5.4 again. Looking at the poset of the equivalence classes shows that there is no train between the set $\{1,2,3\}$ and $\{1,3\}$ (though it is subduction related witnessed by the identity). As an other example consider $\{3,4\}$ and $\{1,2\}$. We know the train: $yz \cdot v$. For the buses see Fig. 8.7. So putting together trip we get $wwwww \cdot yz \cdot v \cdot wwww$.

Using a Coordinate System

We've seen already what knowledge we can get by using the skeleton of an automaton. The next step would be to provide access to these capabilities by the help of the coordinate system of the cascaded product. The hierarchical coordinates for states and transformations can be considered as a nice interface for any possible user (software, robot, human). But this has some difficulties.

- Contrary to our example of adding numbers in decimal notation, in general it is not the case that we have inverses in the holonomy components. We do have constant maps at our disposal, but they may not be usable due to issues with nonimage tiles (see Section 5.7.3).
- If we have a tuple of component actions (which we get after trying to find out what actions should we take in the components in order to achieve the desired state) we have two problems:
  - Distinct cascaded transformations can produce the same actual tuple of component actions, therefore the solution might not be unique.
  - Unlike the cascaded states, not all cascaded transformations have preimages in the division. We may construct a cascaded transformation, for which we do not have a corresponding transformation in the original automaton.

  These problems imply that we end up in a computationally very inconvenient situation, where we have to search the vast space of component action combinations.

Examples

As a positive example, we can mention the residue class rings of modulo a power of 2, since for R_{2^n} the holonomy method gives us the very common binary representation. In this case there are no parallel components and the two states on a level correspond to 0 and 1.

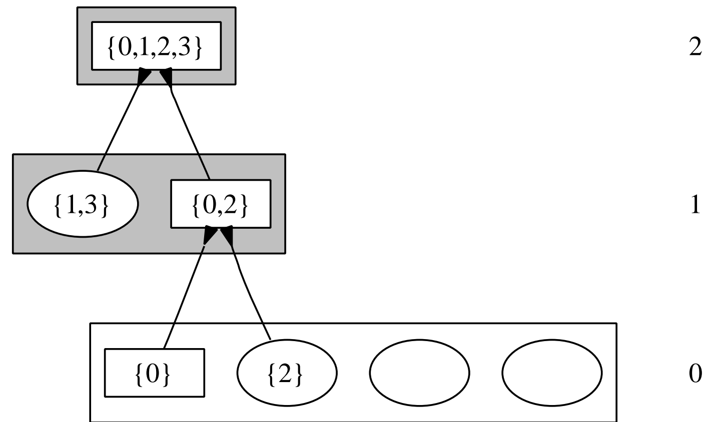


Figure 8.8: The tiling picture of the holonomy decomposition of R_4.

For instance in the case of the decomposition of R_4 we can use the following simplified notation (nb: this is the origin of the binary notation): the states on the top levels are $\{0,2\} = 0$ and $\{1,3\} = 1$, on the first level they are $\{0\} = 0$ and $\{2\} = 1$. The tiling picture is on Fig. 8.8 and the dependency functions are the following.

$\widehat{+1}$:

$$\begin{aligned} f_2^{+1}() &= +1 \\ f_1^{+1}(1) &= +1 \end{aligned}$$

$\widehat{\times 2}$:

$$\begin{aligned} f_2^{\times 2}() &= \text{constant } 0 \\ f_1^{\times 2}(0) &= \text{constant } 0 \\ f_1^{\times 2}(1) &= \text{constant } 1 \end{aligned}$$

$\widehat{\times 3}$:

$$f_1^{\times 3}(1) = +1$$

Knowing this it is easy to solve our problem using coordinates. For example from the cascaded state $(0,1)$ we would like to go to the state $(1,0)$ (from 1

to 2). On the top level we need the operation $+1$, and on the first level also. Looking at the dependency function tables, tha answer is straightforward: we need $\widehat{+1}$. Continuing the search we finds that $\widehat{\times 2}$ is also a good solution, though it uses only constant maps.

Starting from $(0,0)$ and going to the state $(1,0)$ (from 0 to 2) is more problematic, since we cannot find any cascaded transformation from the dependency functions of the generator lifts. Therefore we need to start generating the elements of the wreath product semigroup. In this particular case we know that it will be $\widehat{+1} \cdot \widehat{+1}$, or $\widehat{+1} \cdot \widehat{\times 2}$, but in a more general setup we have no hint where to look for the required transformation. However, this example suggests that if we had the dependency functions explicitly available for all transformations in the cascaded ts, then we could easily look for the required transformations.

For other integers it gives a bit more unusual representation. For instance for R_{18} we have cascaded states in $(\mathbf{2} \times \mathbf{3}) \times (\mathbf{3} \times \mathbf{5}) \times \mathbf{5}$, thus we have to deal with determining in which parallel component we are. In this special case it is still can be done manually (although it is quite cumbersome), since we know a lot about the decomposition of these residue class rings. Moreover, these are unusual representations (not decimal or binary or base n) for rings, and they might be useful for computer algebra systems.

The examples discussed here are quite special and we used a lot of background knowledge of their structure. In the general case we have more complicated problems. But despite the difficulties mentioned here, we firmly believe that this is the way to go for the applications of hierarchical algebraic decompositions in artificial intelligence. Success of the approach is illustrated already for the ring examples.

8.4.3 Capturing Learning

Given any phenomenon, first we obtain a finite description of it. This is not yet understanding, since we need a theory, a coordinatization of the raw data, so we make the hierarchical decomposition. But so far this is only static, whereas the way we get to know the world is a dynamic process. The description of the phenomenon, the observational data might get extended, modified, or becomes more accurate. This change clearly affects the theory itself (and vice versa the theory might give guidelines regarding what data to collect). The growing knowledge forces us to revise or replace existing theories. In this context, by learning we mean exactly this process of changing understanding.

Our search of the space of decompositions in Section 8.2.1 showed that small changes may give very different results, in that case in terms of the number of hierarchical levels, thus clearly results in very different decompositions. For the time being we can conclude that the space of the holonomy decompositions is very rugged, and needs closer inspection guided by more

specific questions.

8.5 Summary

Here we further deepened our understanding of the holonomy decomposition by studying examples of decompositions in a detailed way. We also showed the difficulties of applying the holonomy decomposition method as the formal models of understanding. The results of this chapter give directions for future research.

Chapter 9

Achievements and Future work

9.1 Contributions to Knowledge

At this point, before discussing the possible future directions of this research, we should summarize what has been done in this particular work.

- **Assessing feasibility of computational implementations for the Krohn-Rhodes Theory.** Although it is a very difficult problem to generate algebraic decompositions of finite state automata, we have shown that it is doable by using the computational power available on today's computers. At least, for the time being the software tools can provide valuable raw data for the theoretical investigations of the nature of these hierarchical decompositions.

- **Developing a computational toolkit for the Krohn-Rhodes Theory.** Two different proof technique were implemented evaluated in this work: the $V \cup T$ and the holonomy method. Due to its iterative algorithm the $V \cup T$ method gives such redundant decompositions, which are not usable except in special cases for some theoretical research. Therefore the holonomy method was chosen for a more detailed study and for a more capable computational implementation.

- **Detailed study of the holonomy decomposition.** We have described a constructive proof of the holonomy decomposition in such a detailed way (several explaining lemmas before bigger proofs, simplified notation, etc.), that understanding the main theorem or developing its computational implementations should be easy after reading this proof.

  Writing this version of the proof yielded a small but useful theoretical result: in the case of the holonomy decomposition every cascaded state

is a lift of some state in the original automaton. This was not known before and it makes a proof shorter.

- **New methods for constructing holonomy group components.** During the implementation we developed two different methods (beyond the trivial brute force solutions) for locating the holonomy groups in the characteristic semigroup. One uses techniques from formal language theory, while the other one uses the hierarchical dependency structure of the holonomy decomposition.

- **Visualization of the structure of the holonomy decomposition.** The automatically generated diagrams (tiling picture, tile automata, etc.) provide a very easy way of understanding the inner workings of holonomy decomposition.

- **Initial exploration and key examples.** With the computational tool available it had become possible to do systematic explorations of the vast space of holonomy decompositions.

  With random and guided search we could find interesting long decompositions, we also find a provably sharp upper bound on size of the state set of a holonomy component.

  Our exploration of finite rings of integers modulo n yielded a nice structural theorem for their decompositions.

  We present numerous examples, and they serve two different purposes. First, they demonstrate important features of constructions, definitions, or proofs. Secondly, some examples pinpoint some crucial problems or give good intuitions, which might be in the focus of the future research.

9.2 Possible Future Research Directions

Since there can be many possible research projects utilizing the computational tools for the Krohn-Rhodes Theory, we focus here on those questions of efficient computing of the holonomy decomposition, that have the highest priority.

Efficient partial calculation of $\mathcal{I}$. In Section 7.3.2 we have shown that we do not need the full extended set of images in order to calculate the holonomy decomposition. However the partial calculation of $\mathcal{I}$ is not yet solved due to the problem of nonimage tiles (Section 5.7.3).

Relationship between generator sets. The dependency function based method gives us generator sets for the holonomy components, which are comparable to the original generator set regarding their size. But we have to make the relation more precise.

Working with coordinates. We showed the potential of holonomy decompositions to serve as formal models of understanding (buses and trains in Section 8.4.2), but it is still a question how to use its "interface" efficiently. We need to find effective algorithms for manipulating coordinates on purpose.

More examples. We still have only considered very few examples of fruitful application areas of automated holonomy decompositions.

9.3 Exploring a New Landscape

Philosophy is about questions, not about answers. Any philosophical treatise claiming to have firm answers is suspicious. Science is different, but not entirely. The value of good answers is not doubtful, but good questions pregnant with important answers are also vital for scientific development.

Here we opened up a vast area for further research. As a final phase of developing the Krohn-Rhodes Theory we adapted the holonomy method for a computational implementation. Having a working computational tool is essential for the further development of the theory. Like with telescope or microscope, with the software we can see things which we could not glimpse before. Despite the limitations of the current tool it is already far far beyond the capabilities of the pencil and paper method. Of course, we do not mean replacing the classical mathematical method, we just supply it with more substantial data to work on.

With the computational implementation at hand, now we can ask countless questions such as 'What is the hierarchical structure of this phenomenon or what are the building blocks of that thing?'. We can examine new aspects of well-known structures, as we did in the case of finite integer residue class rings. Or we can attack problems where our knowledge is close to nothing. For instance observing evolutionary development in artificial life systems needs an understanding of how the individuals reuse and change existing components, measuring the complexity of the evolved organisms.

- The holy grail of finite semigroup theory is determining and efficiently computing Krohn-Rhodes Complexity (or group complexity). The computational tools are clearly not for solving this problem, but seeing actual decompositions, knowing the components may help in determining the KR complexity of particular examples.

- As a 'bootstrap' process the computational tool can be used for decomposing well-known structures (as we did in the case of finite residue class rings of integer of modulo n). This might seen to be a 'hide and seek' type game: we hide something in the bush, then happily find it. But meanwhile we can get useful knowledge about how hierarchical structure encodes an understanding of the system.

All we need now is to repeat our starting statement, and after this long trip we made the significance of the sentence should be apparent.

For any finite system a working hierarchical model can be generated automatically.

Appendix A

Decompositions of Finite Residue Class Rings of Integers Modulo n, up to $n = 20$

The holonomy decompositions of first 20 nontrivial residue class rings of of integers $\mathbb{Z}$. Note that the numbers of levels corresponds to the number of prime factors of n with multiplicities. The number of states at the top hierarchical level (rightmost) equals the sum of primes dividing n. It is also worth noting how the top level components are reused in the decomposition of bigger rings. For instance the top level component of the decomposition of R_5 appears in the decompositions of R_{10}, R_{15} and R_{20}.

Ring	Levels	$\lvert\mathcal{I}\rvert$	Holonomy Decomposition
R_2	1	3	$(\mathbf{2}, \overline{S_2})$
R_3	1	4	$(\mathbf{3}, \overline{S_3})$
R_4	2	7	$(\mathbf{2}, \overline{C_2}) \wr (\mathbf{2}, \overline{C_2})$
R_5	1	6	$(\mathbf{5}, \overline{G_{5:4}})$
R_6	2	12	$(\overline{(\mathbf{2}, C_2) \times (\mathbf{3}, S_3)}) \wr (\mathbf{5}, \overline{D_{12}})$
R_7	1	8	$(\mathbf{7}, \overline{G_{7:6}})$
R_8	3	15	$(\mathbf{2}, \overline{S_2}) \wr (\mathbf{2}, \overline{S_2}) \wr (\mathbf{2}, \overline{S_2})$
R_9	2	13	$(\mathbf{3}, \overline{S_3}) \wr (\mathbf{3}, \overline{S_3})$
R_{10}	2	18	$(\overline{(\mathbf{2}, C_2) \times (\mathbf{5}, G_{5:4})}) \wr (\mathbf{7}, \overline{G_{40}})$
R_{11}	1	12	$(\mathbf{11}, \overline{G_{11:10}})$
R_{12}	3	28	$(\overline{(\mathbf{2}, C_2) \times (\mathbf{3}, S_3)}) \wr (\overline{(\mathbf{2}, C_2) \times (\mathbf{5}, D_{12})}) \wr (\mathbf{5}, \overline{D_{12}})$
R_{13}	1	14	$(\mathbf{13}, \overline{G_{13:12}})$
R_{14}	2	24	$(\overline{(\mathbf{2}, C_2) \times (\mathbf{7}, G_{7:6})}) \wr (\mathbf{9}, \overline{G_{84}})$
R_{15}	2	24	$(\overline{(\mathbf{3}, S_3) \times (\mathbf{5}, G_{5:4})}) \wr (\mathbf{8}, \overline{G_{120}})$
R_{16}	4	31	$(\mathbf{2}, \overline{S_2}) \wr (\mathbf{2}, \overline{S_2}) \wr (\mathbf{2}, \overline{S_2}) \wr (\mathbf{2}, \overline{S_2})$
R_{17}	1	18	$(\mathbf{17}, \overline{G_{17:16}})$
R_{18}	3	39	$(\overline{(\mathbf{2}, C_2) \times (\mathbf{3}, S_3)}) \wr (\overline{(\mathbf{3}, S_3) \times (\mathbf{5}, D_{12})}) \wr (\mathbf{5}, \overline{D_{12}})$
R_{19}	1	20	$(\mathbf{19}, \overline{G_{19:18}})$
R_{20}	3	42	$(\overline{(\mathbf{2}, C_2) \times (\mathbf{5}, G_{5:4})}) \wr (\overline{(\mathbf{2}, C_2) \times (\mathbf{7}, G_{40})}) \wr (\mathbf{7}, \overline{G_{40}})$

Appendix B

Dilworth's Theorem

The following very useful theorem establishes an important connection between the width of a partially ordered set and its chain decomposition. A *chain* is a subset of the partially ordered set in which every pair of elements is comparable. An *antichain* is a subset of the partial order in which no two elements are comparable.

Theorem B.1 (Dilworth's Theorem) *Let $P = (A, \leq)$ be a partial order. Then the minimum number of chains needed to cover A equals the maximum number of elements in an antichain.*

The original proof is in [Dil50], and a very short proof can be found in [Tve67]. In the context of directed graphs the theorem is also presented in [BJG02], and in the context of lattices and orders in [DP02].

Appendix C

Related Publications

PUBLICATIONS

1. Attila Egri-Nagy, C. L. Nehaniv: **Cycle Structure in Automata and the Holonomy Decomposition**, Acta Cybernetica, (in press)

2. Attila Egri-Nagy, C. L. Nehaniv: **Algebraic Hierarchical Decomposition of Finite State Automata: Comparison of Implementations for Krohn-Rhodes Theory (poster)** CIAA 2004. Ninth International Conference on Implementation and Application of Automata, LNCS 3317. 315-316.

CONFERENCE PRESENTATIONS

1. **WSA 2005**,Workshop on Semigroups and Automata, Joint Satellite Workshop to ICALP'05 and CSL 2005, Lisbon, July 16th, 2005, **Finite residue class rings of integers modulo n from the viewpoint of global semigroup theory**

2. **CSL2005**, International Conference on Semigroups and Languages In Honour of the 65 th birthday of Donald B. McAlister, July 12,13, 14 and 15, 2005, Lisboa, PORTUGAL, **Exploration of the Space of Finite State Automata Using the Holonomy Decomposition**

3. **BCTCS 2005**, 21st British Colloquium for Theoretical Computer Science, 22-24 March 2005, University of Nottingham, UK, **Algebraic Decompositions of Finite Automata and Formal Models of Understanding**. Abstract published in Bulletin of the European Association for Theoretical Computer Science (EATCS), Number 86, p246., June 2005.

4. **CIAA 2004**, Ninth International Conference on Implementation and Application of Automata, Queen's University, Kingston, Ontario, Canada,

July 22-24, 2004, **Algebraic Hierarchical Decomposition of Finite State Automata: Comparison of Implementations for Krohn-Rhodes Theory**, (poster)

5. CS^2, The Fourth Conference of PhD Students in Computer Science, Szeged, Hungary, July 1-4, 2004, **Holonomy Decomposition of Finite State Automata**

6. **MACS 2004**, 5th Joint Conference on Mathematics and Computer Science June 9-12, 2004, Debrecen, Hungary, **Algebraic Decompositions of Finite State Automata and Formal Models of Understanding**

Appendix D

Software Architecture

D.1 Grasp

The $V \cup T$ method was implemented as an external package for GAP [gap02]. It implements a function which returns the list of the cascaded components of the decomposition of a ts. The iterative implementation of this function is the literal translation of the proof of Lemma 4.1.

D.2 jGrasp

The software tool for implementing the holonomy decomposition has a more complicated architecture and it is a lot more capable. The tool consists of the following interacting packages:

The core system. This part is written in the `Java2` language and it is responsible for the main calculations of the decompositions. It has the following subsystems:

`tsengine` Representations and basic algorithms for handling transformations, words, and semigroups.

`decomposer` Using the functionality of `tsengine` the algorithms for the decompositions are implemented separately. This part contains the algorithms for calculating the image and subduction relations, for constructing the skeleton, for constructing holonomy groups, for recording dependency function entries, and so on.

`io` The input and the output of the system are in text files and this subsystem is responsible for handling these files.

Visualization. The core system generates the definitions of the graphs to be displayed and the actual figures are rendered by `GraphViz` [EGK+03].

Group algorithms. We use GAP for classifying the holonomy group components.

Execution. The execution of the decomposition and generating the output diagrams are coordinated by UNIX shell-scripts.

jGrasp has other functionalities as well (calculating the micro structure of transformations (functional digraphs), exploring one-point mutation neighbourhood of transformation semigroups, etc.), but currently the main functionality is the decomposition of a ts given by its generator transformations. Future development should concentrate on making the decomposition process more interactively accessible, probably by implementing the core system in GAP.

The latest information about the software toolkits described here can be found on the website http://graspermachine.sf.net.

Appendix E

Glossary of Symbols

$\leq, \trianglelefteq, \prec$	subduction, image, tile of relations
$\times, \wr$	direct, wreath products
$\tilde{}$	being a lift of a state or transformation
1	the identity element
1_A	the identity transformation on the set A
$\mathcal{A}, \mathcal{B}$	automata
$\mathcal{A}_Q$	the tile automaton of Q
$\mathcal{B}_P$	the set of tiles of P
$\mathcal{B}_i$	state set of the component on level i
$\mathbf{B}_i$	an element of $\mathcal{B}_i$
A, B	state sets
a, b	states
$(A, S), (B, T)$	transformation semigroups
(A, S^I)	$(A, S \cup \{1_A\})$
$\mathbf{n}$	the set of integers $\{0, 1, \ldots, n-1\}$
C_n	the cyclic group on n points
D_n	the dihedral group with order n
$D(\mathcal{A})$	the state transition graph of $\mathcal{A}$
δ	state transition function
E_Q	union of equivalence classes having elements in $\mathcal{B}_Q$
f_s^i	dependency function of s on level i
$G_{n:k}$	a semidirect product $C_n \rtimes C_k$
G_Q	permutator group of Q
$\mathcal{I}$	the extended set of images
H_Q	holonomy group of Q
$h(P)$	the height of P
$\mathrm{im}(t)$	the image set of transformation t
$\mathcal{L}, \mathcal{R}, \mathcal{J}$	Green's relations
$\overrightarrow{m}_P$	mapping from P to $\overline{P}$
$\overleftarrow{m}_P$	mapping from $\overline{P}$ to P
P, Q	elements of $\mathcal{I}$

S, T	semigroups
s, t	elements of semigroups (transformations)
$\sigma(P, B)$	selector function
$\hat{\sigma}(P,' B)$	inverse selector function
$\mathcal{T}_A, \mathcal{T}_n$	full transformation semigroup on the set A, on $\mathbf{n}$
X, Y	alphabets
X^+, X^*	the free semigroup, monoid on X
x, y, z	input symbols
v, w, u	words
w_{PQ}	a witness for $P \leq Q$
$\sqrt{w}$	the root of word w

Bibliography

[Arb68] Michael A. Arbib, editor. *Algebraic Theory of Machines, Languages, and Semigroups*. Academic Press, 1968.

[Ash56] W. Ross Ashby. *An Introduction to Cybernetics*. Chapman & Hall, London, 1956.

[BJG02] Jørgen Bang-Jensen and Gregory Gutin. *Digraphs – Theory Algorithms and Applications*. Springer, 2002.

[Boo91] Grady Booch. *Object Oriented Analysis and Design with Applications*. Benjamin/Cummings Pub. Co., 1991.

[Bro99] Rodney A. Brooks. *Cambrian Intelligence: The Early History of the New AI*. MIT Press (A Bradford Book), 1999.

[CH91] Sang Cho and Dung T. Huynh. Finite-automaton aperiodicity is PSCPACE-complete. *Theoretical Computer Science*, 88:99–116, 1991.

[CP67] A.H. Clifford and G.B. Preston. *The Algebraic Theory of Semigroups (Mathematical Survey, No 7)*, volume 1 of *Mathematical Survey*. American Mathematical Society, 2nd edition, 1967.

[Dil50] R.P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51:161–166, 1950.

[DLM05] Manuel Delgado, Steve Linton, and Jos Morais. GAP package automata. (`http://cmup.fc.up.pt/cmup/mdelgado/automata/`)., 2005.

[DN05] Pál Dömösi and Chrystopher L. Nehaniv. *Algebraic Theory of Finite Automata Networks: An Introduction*, chapter 3, The Krohn-Rhodes and Holonomy Decomposition Theorems. SIAM Series on Discrete Mathematics and Applications, 2005.

[DP02] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order, Second Edition*. Cambridge University Press, 2002.

[EGK+03] J. Ellson, E.R. Gansner, E. Koutsofios, S.C. North, and G. Woodhull. Graphviz and dynagraph – static and dynamic graph drawing tools. In M. Junger and P. Mutzel, editors, *Graph Drawing Software*, pages 127–148. Springer-Verlag, 2003. `http://www.graphviz.org`.

[Eil76] Samuel Eilenberg. *Automata, Languages and Machines*, volume B. Academic Press, 1976.

[EN02] Gillian Z. Elston and Chrystopher L. Nehaniv. Holonomy embedding of arbitrary stable semigroups. *International Journal of Algebra and Computation*, 12(6):791–810, 2002.

[ENN03] Attila Egri-Nagy and Chrystopher L. Nehaniv. GrasperMachine, Computational Semigroup Theory for Formal Models of Understanding. `(http://graspermachine.sf.net)`., 2003.

[ENN04] Attila Egri-Nagy and Chrystopher L. Nehaniv. 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, 2004.

[Ési00] Zoltán Ésik. A proof of the Krohn-Rhodes decomposition theorem. *Theoretical Computer Science*, 234:287–300, 2000.

[gap02] GAP – Groups, Algorithms, and Programming, Version 4.3. `(http://www.gap-system.org)`., 2002.

[Gin68] Abraham Ginzburg. *Algebraic Theory of Automata*. Academic Press, 1968.

[GLS94] Daniel Gorenstein, Richard Lyons, and Ronald Solomon. *The Classification of Finite Simple Groups*. AMS, 1994.

[Hal59] Marshall Hall. *The Theory of Groups*. The Macmillan Company, New York, 1959.

[HLR88] Karsten Henckell, Susan Lazarus, and John L. Rhodes. Prime decomposition theorem for arbitrary semigroups. *Journal of Pure and Applied Algebra*, 55:121–172, 1988.

[Hof79] Douglas R. Hofstadter. *Gödel, Escher, Bach: an Eternal Golden Brain*. The Harvester Press Ltd, 1979.

[Hol75] John H. Holland. *Adaptation in natural and artificial systems*. The University of Michigan Press, Ann Arbor, 1975.

[Hol82] W. M. L. Holcombe. *Algebraic Automata Theory.* Cambridge University Press, 1982.

[Joh01] Steven Johnson. *Emergence.* Scribner, 2001.

[Knu98] Donald E. Knuth. *The Art of Computer Programming – Sorting and Searching*, volume 3. Addison-Wesley, 2nd edition, 1998.

[KR65] Kenneth Krohn and John Rhodes. Algebraic theory of machines. I. Prime decomposition theorem for finite semigroups and machines. *Transactions of the American Mathematical Society*, 116:450–464, April 1965.

[KRT68] Kenneth Krohn, John L. Rhodes, and Bret R. Tilson. *Algebraic Theory of Machines, Languages, and Semigroups* (M. A. Arbib, ed.), chapter 5, The Prime Decomposition Theorem of the Algebraic Theory of Machines, pages 81–125. Academic Press, 1968.

[KS98] Stanislav Kublanovsky and Mark Sapir. Potential divisibility in finite semigroups is undecidable. *International Journal of Algebra and Computation (IJAC)*, 8(6):671–680, December 1998.

[LJ80] George Lakoff and Mark Johnson. *Metaphors We Live By.* University of Chicago Press, 1980.

[LN00] George Lakoff and Rafael E. Núnez. *Where Mathematics comes from? – How the embodied mind brings mathematics into being.* Basic Books, 2000.

[Lot83] M. Lothaire, editor. *Combinatorics on Words.* Cambridge University Press, 1983.

[MMP+95] O. Matz, A. Miller, A. Potthoff, W. Thomas, and E. Valkema. Report on the program AMoRE. Technical Report 9507, Christian Albrecht University of Kiel, October 1995.

[Neh92] Chrystopher L. Nehaniv. *Global Sequential Coordinates on Semigroups, Automata, and Infinite Groups.* PhD thesis, University of California, Berkeley, 1992.

[Neh94] Chrystopher L. Nehaniv. Algebraic engineering of understanding: Global hierarchical coordinates on computation for the manipulation of data, knowledge, and process. In *Proc. 18th Annual International Computer Software and Applications Conference (COMPSAC 94)*, pages 418–425. IEEE Computer Society Press, 1994.

[Neh96a] Chrystopher L. Nehaniv. Algebra and formal models of understanding. In Masami Ito, editor, *Semigroups, Formal Languages and Computer Systems*, volume 960, pages 145–154. Kyoto Research Institute for Mathematics Sciences, RIMS Kokyuroku, August 1996.

[Neh96b] Chrystopher L. Nehaniv. A simple, direct proof of the Krohn-Rhodes theorem. In Y. Kobayashi and R. Matsuda, editors, *Proceedings of the 20th Symposium on Semigroups, Languages, and Their Related Areas*, pages 29–33, Mito, Japan, November 1996.

[NR00] Chrystopher L. Nehaniv and John L. Rhodes. The evolution and understanding of hierarchical complexity in biology from an algebraic perspective. *Artificial Life*, 6:45–67, 2000.

[Rho71] John L. Rhodes. *Applications of Automata Theory and Algebra with the Mathematical Theory of Complexity to Finite-State Physics, Biology, Philosophy, Games, and Codes.* University of California at Berkeley, Mathematics Library, 1971.

[RW89a] John L. Rhodes and Pascal Weil. Decomposition techniques for finite semigroups using categories, I. *Journal of Pure and Applied Algebra*, 62:269–284, 1989.

[RW89b] John L. Rhodes and Pascal Weil. Decomposition techniques for finite semigroups using categories, II. *Journal of Pure and Applied Algebra*, 62:285–312, 1989.

[Sch65] M. P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8:190–194, 1965.

[Ser03] Ákos Seress. *Permutation Group Algorithms*, volume 152 of *Cambridge Tracts in Mathematics.* Cambridge University Press, 2003.

[Shy01] H. J. Shyr. *Free monoids and languages.* Hon Min Book Company, Taichung, Taiwan, 2001.

[Sim96] Herbert A. Simon. *The Sciences of the Artificial - 3rd Edition.* MIT Press, 1996.

[Ste03] Luc Steels. Intelligence with representation. *Philosophical Transactions: Mathematical, Physical and Engineering Sciences*, 361(1811):2381–2395, 2003.

[Tve67] H. Tverberg. On Dilworth's decomposition theorem for partially ordered sets. *Journal of Combinatorial Theory*, 3:305–306, 1967.

[Zei67] H. Paul Zeiger. Cascade synthesis of finite state machines. *Information and Control*, 10:419–433, 1967. plus erratum.

[Zei68] Paul Zeiger. *Algebraic Theory of Machines, Languages, and Semigroups* (M. A. Arbib, ed.), chapter 4, Cascade Decomposition Using Covers, pages 81–125. Academic Press, 1968.