Proceedings of the 8th International Conference on Applied Informatics
Eger, Hungary, January 27–30, 2010. Vol. 1. pp. 229–237.

On Constructing and Visualizing Cascaded Automata*

Attila Egri-Nagy, Chrystopher L. Nehaniv

Royal Society Wolfson BioComputation Research Lab
Centre for Computer Science and Informatics Research
University of Hertfordshire
Hatfield, Hertfordshire, United Kingdom

Abstract

Cascaded compositions are important constructions for understanding the structure of a given automaton, though they are quite complicated objects (especially their algebraic counterpart, the wreath product). Therefore, despite several attempts to bring them in the mainstream, cascaded structures still play an underappreciated role in computer science. Here we try to help this situation by giving easy definitions, visualization techniques, efficient constructions of cascaded structures based on dependency functions, i.e. hierarchically built automata. These mathematical tools are supported by computational implementations.

1. Introduction

Hierarchical coordinatizations enter automata theory by Krohn-Rhodes theory [8], which states that any finite state automaton can be emulated by a wreath product of its simpler components (permutation automata and flip-flops). One would think that such an important decomposition theory will have a central role in automata theory and in computer science. On the contrary, though there are many possible and very promising applications (for a sweeping view see [10]), there exist only a few actual uses of the coordinatization concept (e.g. [9, 1]) independent of the original line of research. Also, there was no computational implementation until [3], which only recently became available as a computer algebra package [4] suitable for the wider community. So here we would like to address the following issues underlying the difficulties of dealing with cascaded structures:

1. Wreath products have difficult definitions mostly in abstract algebraic terms.

*Partial support for this work by the OPAALS (FP6-034824) project is gratefully acknowledged.

2. Wreath products are huge structures, therefore computing with them is usually not possible due to combinatorial explosions.

3. Due to their complexity, cascaded structures are difficult to grasp (except in some special cases) and thus they do not contribute to our understanding of the original automaton being modelled as a cascade.

We suggest the following solutions:

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

2. Dealing with sub-wreath products.

3. Simplified visual notation for cascaded automata.

Purposefully, we deviate from the usual mathematical presentation style (definitions, theorems and possibly some applications), and use a problem driven exposition.

2. Group Coordinatizations

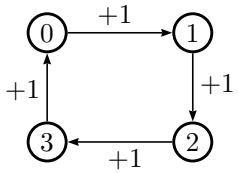
2.1. Counters

2.1.1. Modulo-4 counter

The modulo-4 counter automaton is the most simple example that admits a nontrivial hierarchical coordinatization and still does some interesting calculation.

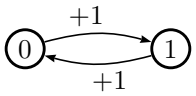
$$\mathcal{C}_4 = \big(\{0,1,2,3\}, \langle +1\rangle = C_4\big)$$

where $+1$ is the counting operation written as (0 1 2 3) in cyclic notation, and C_4 is the cyclic group of order 4.



It is easy to guess that we can coordinatize this automaton by using two modulo-2 counters.

$$\mathcal{C}_2 = \big(\{0,1\}, \langle +1\rangle = C_2\big), \text{ where } +1 = (0\ 1)$$



The +1 symbol is overloaded but it should be clear from the context whether it belongs to C_2 or C_4.

2.1.2. Coordinatizing the state set

The states of the hierarchically constructed modulo-4 counter will have the form

$$(x_1, x_2) \quad x_i \in \{0, 1\} \quad (1 \leq i \leq 2)$$

as the states in a coordinate position come from the state set of the corresponding component $\mathcal{C}_2$. For encoding 4 states with 2 bits can be done by the usual binary notation,

$$\begin{array}{ll} 0 & \mapsto (0,0) \\ 1 & \mapsto (1,0) \\ 2 & \mapsto (0,1) \\ 3 & \mapsto (1,1) \end{array}$$

but note that the order is different. The top level of the hierarchy is the position of ones, the bottom is the position of twos. This may look slightly counterintuitive as the higher powers of 2 indicate bigger (thus more significant) values, but from the point of view of counting, updating position one does not depend on the coordinate values deeper in the hierarchy, but not the other way around. Therefore the position of ones is the top (least dependent) component.

2.1.3. Coordinatizing the counting operation

We would like to bring the operation +1 into a coordinatized format, but it is obvious that we cannot just simply take operations from each component, as +1 does different things at the bottom level depending on the value of the top level, i.e. we need to somehow represent the carry bit. Therefore we need to introduce dependency functions.

We fix a list of components $L = \big((X_1, S_1), \ldots, (X_n, S_n)\big)$, then

Definition 2.1. A *dependency function* d_i in L (of level i) is a function

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

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

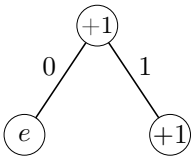
Formally the coordinatized version of $+1$ will be

$$+1 \mapsto (d_1^{+1}, d_2^{+1})$$

where d_1^{+1} is a constant function $\varnothing \to C_2$, and $d_2^{+1} : \{0,1\} \to C_2$ from the state set of the top level to the group component of the second level.

The choice for the top level is straightforward: $+1$ of C_2. For the second level, it is slightly more complicated:

$$\begin{array}{lll} 0 & \mapsto & e \\ 1 & \mapsto & +1 \in C_2 \end{array}$$



where e is the identity. If the top level is 0 then adding one will change only the first position. However, if the top level coordinate is 1, then there is a carry, so we have to advance the deeper 2-counter as well. The tree notation on the right incorporates the dependency functions on both levels.

2.1.4. Using a computer

Using the `SgpDec` [4] software package for the `GAP` [7] computer algebra package, we calculate the coordinatization for $\mathcal{C}_4$. Here we face another recoding of symbols: the state starts with 1 instead of 0, due to computer algebra implementational considerations (the first element of the list is indexed by 1, not 0 as in `C`).

First we calculate the decomposition to find the components of C_4.

```
gap> ld := LagrangeDecomposition(C4);
Lagrange decomposition of:Group([ (1,2,3,4) ])
1 2 Group([ (1,2) ])
2 |-2 Group([ (1,2) ])
```

Coordinatizing the states gives some surprise as the coding seems to be different:

```
gap> for i in [1..4] do Print(i," -> ");
       Display(Raise(ld,i));
     od;
1 -> C(1,1)
2 -> C(2,2)
3 -> C(1,2)
4 -> C(2,1)
```

Also, the dependency function for $+1$ seems to be wrong at first sight:

```
gap> Display(Raise(ld,(1,2,3,4)));

Cascaded  Permutation
Level 1: 0 -> G2
```

```
[] -> (1,2)

Level 2: 2 -> G2
[1] -> (1,2)
[2] -> ()
```

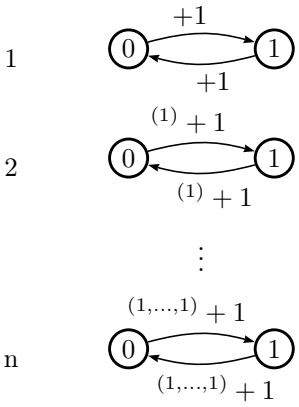
However, closer inspection reveals that the automatic coordinatization is indeed isomorphic to $\mathcal{C}_4$ but it counts in the reverse direction (it looks more like -1). The actual coding used in the decomposition can be changed by choosing different coset representatives [5], but we have to keep in mind that an automatic coordinatization may not be suitable for humans.

2.1.5. Wreath Product Coordinatization

The wreath product contains all dependency functions for each level and all of their possible combinations, hence it keeps exploding combinatorically. We can use the wreath product for coordinatizing $\mathcal{C}_4$, but we get a coordinate system of something bigger: $C_2 \wr C_2 \cong D_8$, which is the group of symmetries of the square. The embedding obviously works $C_4 \hookrightarrow D_8$, but the dihedral group has the flip-symmetry as well, beyond rotations. The discrepancy is relatively big and it gets bigger for groups with more elements. Therefore we aim to have more 'economical' coordinate systems by using sub-wreath products, and in the case of groups we can construct an isomorphic coordinate system.

2.1.6. Generalizing to modulo-2^n counters

It is easy to see how the coordinatization generalizes the modulo-2^n counters: adding 1 on the ith level has no effect except all the digits before are 1s. Following and extending the notation used in [9], we can describe this cascaded structure in a compact way:



The automaton on a particular level defines the corresponding component of the cascade, and the dependency functions are coded in the labels. For example $^{(1,1)}+1$

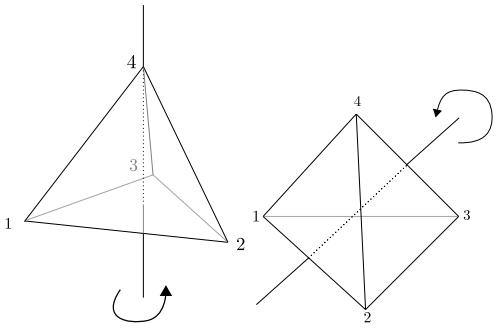


Figure 1: Generator symmetries of the tetrahedron: rotation r and flip f.

reads if we act by the cascaded operation corresponding to the original $+1$ in C_{2^n} and the first two levels both contain 1 as coordinate value, then the labelled transition in the third component should be executed. Important to note that for this notation we simply forget about the labels of the components's own state transitions.

2.2. Rotational Symmetries of the Tetrahedron

With generators $r = (1\ 2\ 3), f = (1\ 2)(3\ 4)$ (rotations of the base triangle, and flipping through an axis through two edges) Fig. 1, A_4 the symmetry group of the tetrahedron admits the following coordinatization:

$$A_4 \cong C_3 \wr_{\mathcal{L}} (C_2 \times C_2).$$

The subscript for the wreath product sign indicates that this is a coordinatization obtained by the Lagrange decomposition algorithm [5]. This algorithm constructs the components according to a subgroup chain (starting from the whole group) by using the action of the corresponding element in the chain on the cosets taken by the next element in the chain. The subgroup chain used here is $A_4 \geq C_2 \times C_2 \geq 1$. Therefore the top component is $C_3 = A_4/(C_2 \times C_2)$. Though it is tempting to think that this component is identical to a subgroup of A_4, for instance $\langle r \rangle$, it is not exactly the case in a direct way, we should interpret the component with respect to its action on the cosets:

$$\begin{aligned} 1 &= \{1, f, rfr^{-1}, r^{-1}fr\} \\ 2 &= \{r, rf, fr, frf\} \\ 3 &= \{r^{-1}, rfr, r^{-1}f, fr^{-1}\} \end{aligned}$$

where these states correspond to number of rotations we make, to $1, r$ and $r^2 = r^{-1}$ respectively. So the first coordinate counts how many rotations were made

regardless the number of flips. The points of the next level are coded in the following way:

$$\begin{aligned} 1 &= \{1\} \\ 2 &= \{f\} \\ 3 &= \{rfr^{-1}\} \\ 4 &= \{r^{-1}fr = r^2fr^{-2}\} \end{aligned}$$

It is easy to recognize that the four states corresponds to the flips around the 3 axes through midpoints of opposite edges, and the identity. Therefore the coordinatization of the tetrahedron along the chief series of its symmetry group is

top level: number of rotations of the base triangle,
bottom level: the axis around which a flip was made.

Thus 'solving the tetrahedron', i.e. getting to a base position, according to this coordinate system would be first rotating the base triangle to the right position then do a flip if needed. In this very small example the solution is very simple, but in more complicated systems, like the Rubik's Cube, the different solving strategies, algorithms can be described by the different coordinatizations.

3. Semigroup Example

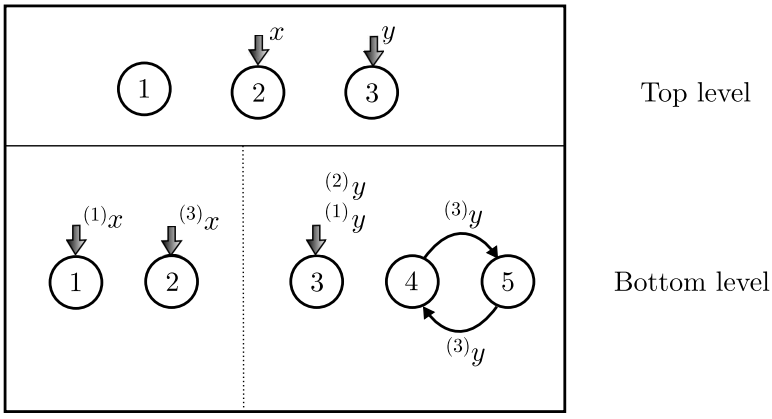


Figure 2: The holonomy decomposition of the transformation semigroup generated by x and y (for details see text).

We now coordinatize a 4 element transformation semigroup S generated by the transformations $x = \left(\begin{smallmatrix}1&2&3&4&5\\2&2&3&3&3\end{smallmatrix}\right)$ and $y = \left(\begin{smallmatrix}1&2&3&4&5\\3&3&3&5&4\end{smallmatrix}\right)$, Fig. 2. For the decomposition we use the holonomy algorithm [6, 2] that gives us permutation-reset components (groups augmented with constant maps). For these special type of cascaded

structures we use another simplification for visualizing: since constant maps are abundant in the resulting cascade they are denoted by a special type of 'fat' arrow, meaning that there is an arrow with the same label from each state within the component to the pointed state. This economical notation makes it easier to recognize permutation components, like C_2 generated by $(3)y$ on the second level, in the second component (Fig. 2). Also, the vertical division indicates 'parallel' components, since distinct components not being in some hierarchical relationship can occupy the same hierarchical level.

The main advantage of this notation is that it shows the components as automata and the connection network simultaneously.

References

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

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

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

[4] Attila Egri-Nagy and Chrystopher L. Nehaniv. *SgpDec – software package for hierarchical coordinatization of groups and semigroups, implemented in the GAP computer algebra system, Version 0.4.86*, 2008-2010. http://sgpdec.sf.nethttp://sgpdec.sf.net.

[5] Attila Egri-Nagy and Chrystopher L. Nehaniv. Subgroup chains and Lagrange coordinatizations of finite permutation groups. http://arxiv.org/abs/0911.5433arXiv:0911.5433v1 [math.GR], 2009.

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

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

[8] Kenneth Krohn, John L. Rhodes, and Bret R. Tilson. The prime decomposition theorem of the algebraic theory of machines. In Michael A. Arbib, editor, *Algebraic Theory of Machines, Languages, and Semigroups*, chapter 5, pages 81–125. Academic Press, 1968.

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

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

Attila Egri-Nagy, Chrystopher L. Nehaniv
Royal Society Wolfson BioComputation Research Lab
Centre for Computer Science and Informatics Research
University of Hertfordshire
Hatfield, Hertfordshire, United Kingdom
e-mail: {A.Egri-Nagy,C.L.Nehaniv}@herts.ac.uk