SGPDEC: CASCADE (DE)COMPOSITIONS OF FINITE TRANSFORMATION SEMIGROUPS AND PERMUTATION GROUPS

ATTILA EGRI-NAGY[1,3], JAMES D. MITCHELL[2], AND CHRYSTOPHER L. NEHANIV[3]

ABSTRACT. We describe how the SGPDEC computer algebra package can be used for composing and decomposing permutation groups and transformation semigroups hierarchically by directly constructing substructures of wreath products, the so called cascade products. transformation semigroup, permutation group, wreath product, Krohn-Rhodes Theory

1. INTRODUCTION

Wreath products are widely used theoretical constructions in group and semigroup theory whenever one needs to build a composite structure with hierarchical relations between the building blocks. However, from a computational and engineering perspective they are less useful since wreath products are subject to combinatorial explosions and we are often interested only in substructures of them. Cascade products precisely build these substructures by defining the hierarchical connections explicitly. As input, given a group or a semigroup with unknown internal structure, the goal of cascade decomposition algorithms is to come up with a list of simpler building blocks and put them together in a cascade product, which realizes in some sense the original group or semigroup. Roughly speaking, for permutation groups, cascade product decompositions can be interpreted as putting the inner workings of the Schreier-Sims algorithm (generalized to any subgroup chain) into an external product form, therefore one can build cascade products isomorphic to the group being decomposed. For semigroups, Krohn-Rhodes decompositions [14] can be computationally represented by cascade products of transformation semigroups.

In this paper we describe how the GAP [9] package SGPDEC [6] implements cascade products and decomposition algorithms and we also give a few simple example computations. This description of the package only focuses on the core functionality of the package.

A *transformation* is a function $f : X \to X$ from a set to itself, and a *transformation semigroup* (X, S) of degree n is a collection S of transformations of X closed under function composition, $|X| = n$. In case S is a group of permutations of X, we call (X, S) a *permutation group*. Using automata theory terminology sometimes we call X the *state set*, often represented as a set of integers $\mathbf{n} = \{0, \ldots, n-1\}$. We write x^s to denote the new state resulting from applying a transformation $s \in S$ to a state $x \in X$.

2. CASCADE PRODUCT BY A MOTIVATING EXAMPLE

To motivate the definition of the cascade product, we consider how the mod-4 counter, the cyclic permutation group $(\mathbf{4}, \mathbb{Z}_4)$, can be constructed from two mod-2 counters. The direct product $\mathbb{Z}_2 \times \mathbb{Z}_2$ contains no element of order 4. Since $\text{Aut}(\mathbb{Z}_2)$ is trivial there is only one semidirect product of $\mathbb{Z}_2$ and $\mathbb{Z}_2$, which equals their direct product. Their wreath product, $\mathbb{Z}_2 \wr \mathbb{Z}_2 \cong D_4$, the dihedral group of the square can be used to emulate a mod-4 counter, since $\mathbb{Z}_4 \hookrightarrow D_4$. But this construction is not efficient, beyond the required rotations the dihedral group has the flip-symmetry as well, doubling the size of the group. However, we would like to have a product construction that is isomorphic to $(\mathbf{4}, \mathbb{Z}_4)$.

This motivates the definition of *cascade products*: efficient constructions of substructures of wreath products, induced by explicit dependency functions [5]. Essentially, cascade products are transformation semigroups glued together by functions in a hierarchical tree. More precisely, let $\big((X_1, S_1), \ldots, (X_n, S_n)\big)$ be a fixed list of transformation semigroups, and dependency functions of the form

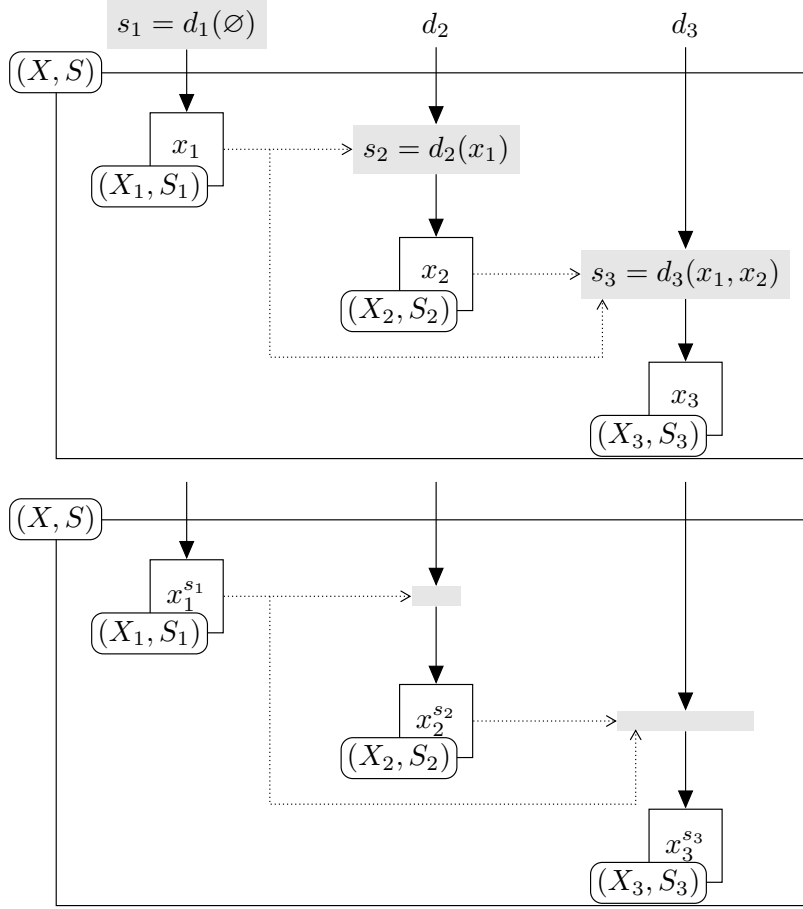$$d_i : X_1 \times \ldots \times X_{i-1} \to S_i, \quad \text{for } i \in \{1, \ldots, n\}.$$

FIGURE 1. Action in a cascade product of components $[(X_1, S_1), (X_2, S_2), (X_3, S_3)]$. The current state (x_1, x_2, x_3) (top) is transformed to the new state $(x_1^{s_1}, x_2^{s_2}, x_3^{s_3})$ (bottom) by the transformation cascade (d_1, d_2, d_3). The component actions s_i are calculated by evaluating the dependency functions of (d_1, d_2, d_3) on the states of the components above. The evaluations are highlighted and they happen at the same time. The dependencies, where the state information travels, are denoted by dotted lines.

A *transformation cascade* is then defined to be an n-tuple of dependency functions $(d_1, \ldots, d_n)$, where d_i is a dependency function of level i. If no confusion arises, on the top level we can simply write $d_1 \in S_1$ instead of $d_1(\varnothing) \in S_1$. The cascade action is defined coordinatewise by $x_i^{d_i(x_1, \ldots, x_{i-1})}$, applying the results of the evaluated dependency functions (see Fig. 1), so that the cascade product can be regarded as a special transformation representation on the set $X_1 \times \ldots \times X_n$. The hierarchical structure allows us to conveniently distribute computation among the components (X_i, S_i), and perform abstractions and approximations of the system modelled as a cascade product. Then if W is a set of transformation cascades $(X_1, S_1) \wr_W \cdots \wr_W (X_n, S_n)$ denotes the transformation semigroup $(X_1 \times \cdots X_n, \langle W \rangle)$, where $\langle W \rangle$ is the semigroup of transformation cascades generated by W.

We can construct $(\mathbf{4}, \mathbb{Z}_4)$ exactly by using two copies of $(\mathbf{2}, \mathbb{Z}_2)$. The generator set contains only one permutation cascade $W = \{(+1, c)\}$, where $+1$ is the generator of $\mathbb{Z}_2$ and c is a dependency function mapping $\mathbf{2}$ to $\mathbb{Z}_2$ with $c(0) = 1_{\mathbb{Z}_2}$, and $c(1) = +1$. The first dependency is a constant (increment modulo 2) while the second dependency implements the carry. Therefore, with fewer dependencies than required by the wreath product, the mod-4 counter can be realized by an isomorphic cascade product: $(\mathbf{2}, \mathbb{Z}_2) \wr_W (\mathbf{2}, \mathbb{Z}_2) \cong (\mathbf{4}, \mathbb{Z}_4)$, see Fig. 2.

$1_{\mathbb{Z}_2}$ $1_{\mathbb{Z}_2}$

+1

(0)   (1)   top

+1

$c(0)$ $c(1)$

$1_{\mathbb{Z}_2}$ +1 $1_{\mathbb{Z}_2}$
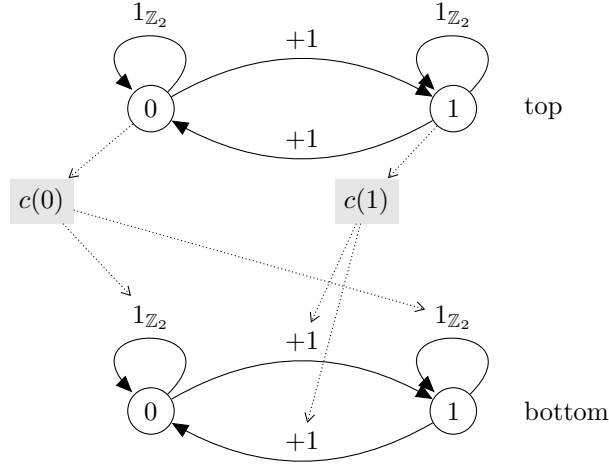
(0)   (1)   bottom

+1

FIGURE 2. Two mod-2 counters cascaded together to build a mod-4 counter.

An immediate consequence of the generality of the cascade product is that several well-known constructions are special cases of the cascade product, and as such they are easy to implement. Direct products consist of all $d = (d_1, \ldots, d_n)$ with each d_i constant. Wreath products consist of all possible dependency functions. Direct, cascade, and wreath products constructions for transformation semigroups are now available in SGPDEC, and iterated wreath products for permutation groups also became a bit more convenient to define.

3. FUNCTIONALITY

There are two different basic ways of using the SGPDEC package. Depending on whether the starting point is a complex structure or a set of (simple) building blocks, we can do *decomposition* or *composition*.

3.1. Composition and Construction.
The questions we aim to answer by constructing cascade products can be of the following types.

(1) What is the (semi)group generated by a given set of transformation cascades?
(2) What can be built from a given set of (simple) components?

The usual scenario is that for a list of components we give a set of cascades as a generating set. For instance, the quaternion group $Q = \langle i, j \rangle$ is not a semidirect product, but it embeds into the full cascade product $(\mathbf{2}, \mathbb{Z}_2) \wr (\mathbf{2}, \mathbb{Z}_2) \wr (\mathbf{2}, \mathbb{Z}_2)$, a group with 128 elements. Therefore, it can be built from copies of $\mathbb{Z}_2$. The dependency functions can only have two values, thus to define cascade permutations it is enough to give only those arguments that give $+1$ (the generator of $\mathbb{Z}_2$). A cascade permutation realizing i is defined by the dependency functions (d_1, d_2, d_3) where $d_2(0) = d_2(1) = d_3(0,0) = d_3(1,1) = +1$ and all other arguments map to the identity. Similarly, a cascade realizing j is defined by (d_1', d_2', d_3') where $d_1'(\varnothing) = d_3'(0,0) = d_3'(0,1) = +1$, (see Fig. 3, note that the state values are shifted by 1). One can check that these two order 4 elements generate the 8-element quaternion group Q. Therefore by $W = \{(d_1, d_2, d_3), (d_1', d_2', d_3')\}$ we have

$$(Q, Q) \cong (\mathbf{2}, \mathbb{Z}_2) \wr_W (\mathbf{2}, \mathbb{Z}_2) \wr_W (\mathbf{2}, \mathbb{Z}_2).$$
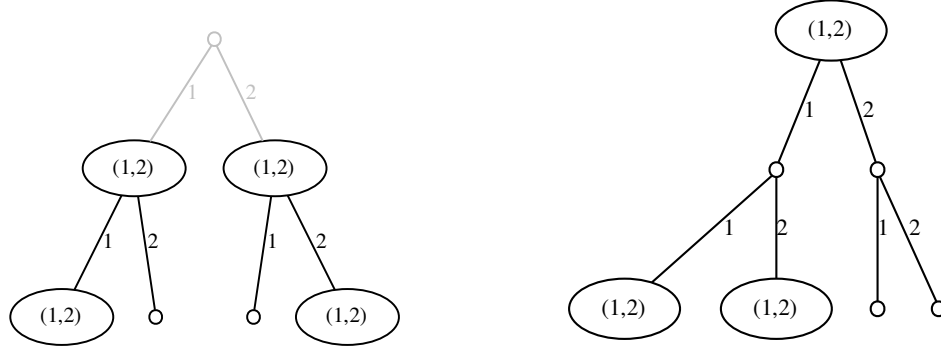
FIGURE 3. Generators of a cascade representation of the quaternion group in a tree form. The edge labels are states, while the nodes contain the action. Empty node corresponds to the identity. The gray part of the tree is fixed.

```
[\small]
gap> Z2:=CyclicGroup(IsPermGroup,2);
Group([ (1,2) ])
gap> d:=Cascade([Z2,Z2,Z2],[[[1],(1,2)],[[2],(1,2)],
                            [[1,1],(1,2)],[[2,2],(1,2)]]);
<perm cascade with 3 levels with (2, 2, 2) pts, 4 dependencies>
gap> dprime:=Cascade([Z2,Z2,Z2],[[[],(1,2)],[[1,1],(1,2)],[[1,2],(1,2)]]);
<perm cascade with 3 levels with (2, 2, 2) pts, 3 dependencies>
gap> StructureDescription(Group([d,dprime]));
"Q8"
```

3.2. Decomposition.

(1) What are the basic building blocks of a given (semi)group?
(2) How can we represent it as a cascade product?

A typical scenario is that for a given composite semigroup or group we choose a decomposition algorithm which returns a cascade product.

3.2.1. *Frobenius-Lagrange Decomposition.* In the case of groups the decomposition uses the idea behind induction in representation theory (see e.g. [1]), so it traces back to Frobenius. Indeed, a special case of them comprises the well-known Krasner-Kaloujnine embeddings [13]. All we need here is just standard group theory, namely the action on cosets, hence the name *Frobenius-Lagrange Decomposition.*

How would someone come up with the generators cascades of the quaternion group in Section 3.1? The easiest solution is to use this group decomposition.

```
[\small]
gap> Q := QuaternionGroup(IsPermGroup,8);
Group([ (1,5,3,7)(2,8,4,6), (1,2,3,4)(5,6,7,8) ])
gap> CQ := FLCascadeGroup(Q);
<cascade group with 2 generators, 3 levels with (2, 2, 2) pts>
```

The actual implementation takes a subgroup chain as input (chief series by default) and form the components by examining the coset space actions derived from the chain. Therefore, the decomposition method can be considered as a generalized Schreier-Sims algorithm [12].

Coordinatewise calculation in a cascade product can also be thought of as a sequence of refining approximate solutions. For instance, each completed step of an algorithm for solving the Rubik's Cube

corresponds to calculating the desired value at a hierarchical level of some cascade product representation and it gives a configuration 'closer' to the solved state.

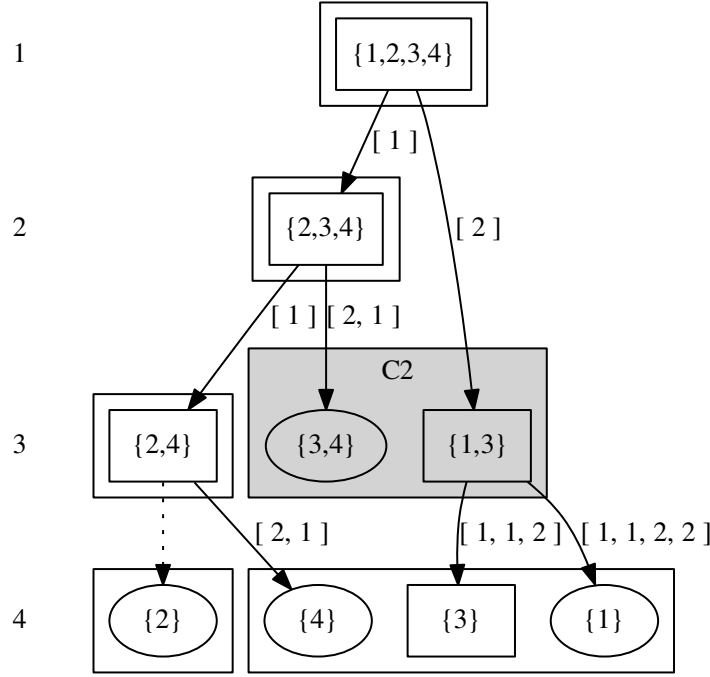FIGURE 4. Tiling picture – the internal details of the holonomy decomposition of the transformation semigroup T generated by t_1 and t_2. The numbers on the left denote the hierarchical levels (level 4 consists of singleton sets and it is needed by the holonomy algorithm but not a component of the cascade product). Outer boxes contain subsets that are mutually reachable from each other under the semigroup action. The arrows indicate how a subset is 'tiled' by its subsets, the arrow labels contain words (sequences of generators) that take a subset to one of its tiles. Dotted arrow means the tile is not an image. Roughly, the holonomy algorithm finds the components by checking the action of the semigroup on a set of tiles.

3.2.2. *Holonomy Decomposition.* For transformation semigroups the holonomy method [4,7,10,11,15–17] is used. The holonomy decomposition works by a close analysis of how the semigroup acts on those subsets of the state set which are images of the state set. As a small example let's define T as the transformation semigroup generated by $t_1 = \left(\begin{smallmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 4 & 4 \end{smallmatrix}\right)$ and $t_2 = \left(\begin{smallmatrix} 1 & 2 & 3 & 4 \\ 3 & 3 & 1 & 3 \end{smallmatrix}\right)$. Calculating its holonomy decomposition and displaying some information can be done by the following commands:

```
[\small]
gap> T:=Semigroup([Transformation([3,2,4,4]),Transformation([3,3,1,3])]);
<transformation semigroup on 4 pts with 2 generators>
gap> HT := HolonomyCascadeSemigroup(T);
<cascade semigroup with 2 generators, 3 levels with (2, 2, 4) pts>
gap> DisplayHolonomyComponents(SkeletonOf(HT));
1: 2
2: 2
3: (2,C2) 2
```

The displayed information tells us that this 13-element semigroup can be realized as the cascade product of four copies of the transformation monoid of constant maps of two points and one instance of $\mathbb{Z}_2$. The components are put together in a 3-level cascade product.

Holonomy decompositions are useful whenever a finite state-transition model of some process needs to be analyzed (e.g. [3]).

3.3. **Visualization.** SGPDEC uses GRAPHVIZ, a widely used graph drawing package [8], for visualisation purposes. The underlying idea is that a function generates source code in the `dot` language for the given mathematical object. Then the actual figure can be generated separately to be included in papers, or using the VIZ package [2] immediately displayed on screen from the GAP command line. Figure 3 and 4 were both auto-generated using GRAPHVIZ.

ACKNOWLEDGMENT

REFERENCES

[1] J. L. Alperin and Rowen B. Bell. *Groups and Representations*. Springer, 1995.

[2] Manuel Delgado, Attila Egri-Nagy, James D. Mitchell, and Markus Pfeiffer. VIZ – *GAP package for visualisation*, 2014. `https://bitbucket.org/james-d-mitchell/viz`.

[3] Paolo Dini, Chrystopher L. Nehaniv, Attila Egri-Nagy, and Maria J. Schilstra. Exploring the concept of interaction computing through the discrete algebraic analysis of the Belousov-Zhabotinsky reaction. *Biosystems*, 112(2):145 – 162, 2013. Selected papers from the 9th International Conference on Information Processing in Cells and Tissues.

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

[5] Attila Egri-Nagy and Chrystopher L. Nehaniv. Cascade Product of Permutation Groups. arXiv:1303.0091v3 [math.GR], 2013.

[6] Attila Egri-Nagy, Chrystopher L. Nehaniv, and James D. Mitchell. SGPDEC – *software package for hierarchical decompositions and coordinate systems, Version 0.7+*, 2014. `http://sgpdec.sf.net`.

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

[8] John Ellson, Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, and Gordon Woodhull. Graphviz and dynagraph static and dynamic graph drawing tools. In *GRAPH DRAWING SOFTWARE*, pages 127–148. Springer-Verlag, 2003.

[9] The GAP Group. GAP – *Groups, Algorithms, and Programming, Version 4.7.5*, 2014. `www.gap-system.org`.

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

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

[12] Derek Holt, Bettina Eick, and Eamonn O'Brien. *Handbook of Computational Group Theory*. CRC Press, 2005.

[13] Marc Krasner and Leo Kaloujnine. Produit complet des groupes de permutations et problème d' extension de groupes. *Acta Scientiarum Mathematicarum (Szeged)*, 14:39–66, 1951.

[14] 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.

[15] Charles Wells. A Krohn-Rhodes theorem for categories. *Journal of Algebra*, 64:37–45, 1980.

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

[17] H. Paul Zeiger. Cascade Decomposition Using Covers. In Michael A. Arbib, editor, *Algebraic Theory of Machines, Languages, and Semigroups*, chapter 4, pages 55–80. Academic Press, 1968.

Centre for Research in Mathematics, School of Computing, Engineering and Mathematics, University of Western Sydney, Australia, and School of Mathematics and Statistics, University of St Andrews, United Kingdom, and Centre for Computer Science & Informatics Research, University of Hertfordshire, United Kingdom

E-mail address: `A.Egri-Nagy@uws.edu.au,jdm3@st-and.ac.uk,C.L.Nehaniv@herts.ac.uk`