Algebraic Hierarchical Decomposition of Finite State Automata: Comparison of Implementations for Krohn-Rhodes Theory

Attila Egri-Nagy and Chrystopher L. Nehaniv

University of Hertfordshire, School of Computer Science
College Lane, Hatfield, Herts AL10 9AB, United Kingdom
{A.Nagy | C.L.Nehaniv}@herts.ac.uk

The hierarchical algebraic decomposition of finite state automata (Krohn-Rhodes Theory) has been a mathematical theory without any computational implementations until the present paper, although several possible and promising practical applications such as automated object-oriented programming in software development [5], formal methods for understanding in artificial intelligence [6], a widely applicable integer-valued complexity measure [8,7], have been described. As a remedy for the situation, our new implementation, described here, is freely available [2] as open-source software. We also present two different computer algebraic implementations of the Krohn-Rhodes decomposition, the $V \cup T$ and holonomy decompositions [4,3], and compare their efficiency in terms of the number of hierarchical levels in the resulting cascade decompositions.

The difficulties of computational implementations of the Krohn-Rhodes decomposition come from the fact that mathematical proofs do not consider computational feasibility, i.e. the space and time complexity of the required calculations. This problem is especially acute in semigroup theory, where semigroups have so many elements. We represent a semigroup by a set of generators (the transformations induced by the input symbols of the automaton) instead of Cayley-table, finite presentation, or explicit enumeration of all elements; transformations are represented as mappings on the set $\mathbf{n} = \{1, \ldots, n\}$. This internal 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 time linear in the number of elements in X. As usual, for getting fast set operations, subsets are represented as bitvectors encoding characteristic functions. For deciding whether element is contained in a set or not, hashtables are used.

Two different decompositions have been implemented in this work. The $V \cup T$ technique and the holonomy decomposition were chosen since they are inherently different, representing distinct classes of algorithms, and their proofs are close to an algorithmic description. The $V \cup T$ method is one of the earliest proof techniques [4]. It works with semigroups and uses the right regular representation for the resulting cascaded components. The main idea of the algorithm is that we iteratively decompose the semigroup into two possibly overlapping subsemi-

groups (a left-ideal and a proper subsemigroup). The iteration ends when the components are left-simple or cyclic semigroups. The list of the components in order form the cascaded product. The 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. Therefore the standard $V \cup T$ technique cannot be used for practical purposes: due to its redundancy it may produce even more components than the order of the characteristic semigroup (e.g. the full transformation on 4 points has $4^4 = 256$ elements and its decomposition has 401 components). Getting more elements than n^n for an automaton with n states is far from being efficient. We implemented the $V \cup T$ method as a package for `GAP` [1].

The holonomy method works by the detailed study of how the characteristic monoid of an automaton acts on the automaton's state set. It looks for and cascades holonomy groups, i.e. subgroups of the syntactic monoid permuting certain sets of subsets of the state set. Isomorphic holonomy groups may be represented in one equivalence class thus avoiding repetitions in the wreath product. Therefore the holonomy algorithm was chosen and further optimized by using a more direct constructive method for holonomy groups instead of brute force breadth-first search based implementation. Due to the experimental nature of the method, it was implemented as standalone software [2].

References

1. GAP – Groups, Algorithms, and Programming, a system for computational discrete algebra Version 4.3. `(http://www.gap-system.org)`., 2002.
2. Attila Egri-Nagy and Chrystopher L. Nehaniv. GrasperMachine, Computational Semigroup Theory for Formal Models of Understanding, experimental software packages. `(http://graspermachine.sf.net)`., 2003.
3. Samuel Eilenberg. *Automata, Languages and Machines*, volume B. Academic Press, 1976.
4. 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.
5. 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.
6. 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.
7. 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.
8. John L. Rhodes. *Applications of Automata Theory and Algebra via the Mathematical Theory of Complexity to Finite-State Physics, Biology, Philosophy, Games, and Codes*. (Univ. California Berkeley Math Library 1971), unpublished book.