The GAP package *automata*

M. Delgado, S. Linton and J. Morais

Abstract

This is a brief description of the GAP package *automata*, a package designed to work with finite state automata and with rational languages.

1 Some history

The package *automata* [DLM05] started with an attempt to implement, in GAP [GAP04] (GAP 3, at the time), an algorithm coming from semigroup theory, namely an algorithm to compute the abelian kernel of a finite monoid [Del98]. It involved, in particular, the computation of a rational expression for the language $\varphi^{-1}(x)$ where $\varphi : A^* \to M$ is a surjective homomorphism from the free monoid A^* onto the finite monoid M. Notice that the language $\varphi^{-1}(x)$ is rational, since it is recognized by M. The desired expression could be obtained by considering the right Cayley graph of M as an automaton and by taking the neutral element of M for the initial state and x to final the state. Although there were software available to compute with automata, the implementation in GAP seemed to be appropriate, since the interaction of this software with software of related areas, in particular for semigroups, would then be easier.

As the size of a rational expression for the language recognized by an automaton grows exponentially with the number of states of the automaton (see, for instance, [Del01]) and for the problem in question one had to handle the expression obtained, it could rapidly be concluded that a better algorithm was needed. Nevertheless, M. Delgado did not give up with this project and, after getting the collaboration first of J. Morais and then of S. Linton, the first public version of the package *automata* became publicly available and accepted by GAP.

We have to refer the great influence of the program *AMoRE* [MMP+95] (which stands for *Automata, Monoids and Regular Expressions*) on all this work. Most of the features of our package can also be found in *AMoRE*. Of course, we could left out the part concerning monoids, since GAP can handle them. (This makes evident a great feature of GAP: one can take advantage of having in the same system implementations of algorithms of related areas.)

In next section we talk briefly about the most relevant functions of this package. Another section is then devoted to functions whose implementations are not based on standard algorithms and we finalize with some words concerning with a package that intends to turn GAP more user-friendly, at least for semigroup theorists.

2 Implemented functions

The *automata* package, on its 1.06 version, has defined about 80 functions. We will now list some of the most relevant ones.

To begin with, there are functions to define an automaton (including a function to generate a random automaton), test its type (deterministic, non deterministic or with ϵ-transitions), test if it is a dense automaton (complete), test whether it is an inverse automaton, a permutation automaton or a reversible automaton and a function to test if a word (given as a string) is recognized by an automaton. Then, there are functions to make an automaton complete, to list the sink states of an automaton, to remove these states, to normalize an automaton (in the sense that the initial states are the lower numbered states and the accepting states the higher numbered ones), to reverse its transitions, to apply a permutation to its states and to return a list of the automata obtained by the application of all the permutations of its states. There are also functions to return the accessible states of an automaton and the useful automaton (the subautomaton defined by the states that are reachable from an initial state).

Two automata are said to be *equivalent* if they recognize the same language. Using the package *automata* one may convert an ϵ-transition automaton into an equivalent non deterministic one without ϵ-transitions and to convert one of this type into an equivalent deterministic one. Additionally, there are functions in *automata* to compute the minimal complete deterministic equivalent automaton to one given, to test if two automata are equivalent and to compute the product automaton of two automata.

Since there is a close relation between automata and digraphs, there are also some functions to deal with them, namely functions to generate a random digraph, to compute the *in* and *out* degree of a vertex, to compute the connected and strongly connected components of a digraph, to reverse its edges, to compute the underlying digraph of an automaton and to convert a digraph into a relation, which is an object handled by GAP.

With respect to rational expressions (RE), there are functions to specify them (including a function to generate a random one over a given alphabet) and to compute the product or union of two REs as well as to compute the star (Kleene operator) of a RE. There are also functions to test whether a RE represents the empty language or the full language (Σ^*, where Σ is the alphabet of the RE in question), as well as functions to test if the language represented by one RE is contained in the language represented by another RE and to test if the languages represented by two REs are disjoint or equal. There are also functions to convert a RE into an automaton that recognizes the language represented by the given rational expression and a function to do the reverse operation.

In relation with group theory there are functions to perform the bijective correspondence between inverse automata and finitely generated subgroups of a free group. From an inverse automaton to a subgroup of the free group one may go through the computation of a geodesic tree and from a subgroup of the free group one may construct an inverse automaton via Stallings foldings (see, for instance, [KM02]).

In relation with semigroup theory there are functions to compute the transition semigroup of a given automaton. Then, the machinery already available in GAP can be used.

Since it is quite convenient to have drawings of automata, we provide in this package a function to perform these drawings, but it depends on an external tool: the *graphviz* [DEG+02] program to visualize graphs. Furthermore, in order to allow the user to give an automaton in a comfortable way, a Tcl/Tk program can be used. This Tcl/Tk program is part of the deposited GAP package *sgpviz* [DM05b] which will be also referred later in Section 4.

3 Some non standard algorithms

Most of the implemented algorithms to perform the tasks referred in the above section are standard, but there are some exceptions. We talk about

some of these in the following subsections.

3.1 Small rational expressions

In this section we shortly describe an heuristic used in the function to convert an automaton into an RE recognizing the same language which was the subject of [DM05a].

When converting an automaton into an equivalent RE we were concerned with the size of the resulting expression. To do this conversion we use the state elimination algorithm and, having noticed that different orders of vertex removal may lead to expressions of different sizes, an heuristic to try to remove the vertices in an order that leads to an expression of "small" size has been attempted. We define the *size* of a regular expression as the number of occurrences of alphabetical symbols in it, including ϵ. In [Del01] the reader may find a brief description of the state elimination algorithm as well as a more formal definition of *generalized transition graph* (GTG), which is similar to that of a finite automaton, but the edges are labeled with regular expressions instead of just letters.

We define the *weight of a* GTG as the sum of the sizes of all regular expressions labeling the edges of the GTG. The *weight of a vertex* is defined as the weight that will be added to the weight of the GTG by the removal of that vertex. Using the convention $\sum_{k=1}^{0} a_k = 0$, one may easily check that the weight of a vertex x can be computed by the formula

$$\sum_{k=1}^{In}(W_{in}(k) \times (Out-1)) \; + \; \sum_{k=1}^{Out}(W_{out}(k) \times (In-1)) \; + \; W_{loop} \times (In \times Out - 1)$$

where In is the number of edges (not loops) that go into x, Out is the number of edges (not loops) that go out from x, $W_{in}(k)$ is the size of the label of the k^{th} edge that goes into x, $W_{out}(k)$ is the size of the label of the k^{th} edge that goes out from x and W_{loop} is the size of the label of the loop around x.

Our approach to the problem of computing small expressions is to remove, at each step, one of the least weighted vertices. Notice that using the above formula, the time consumed to compute the weight of a vertex is not relevant.

The following table, produced using automata obtained from Cayley graphs of certain transformation monoids, shows that the usage of this heuristic gives quite satisfactory results. (The time is measured in GAP units, in a Pentium IV 2.6 GHz.)

Automaton	States	Alph.	Exp. without heuristic		Exp. with heuristic	
			size	time	size	time
$Min(\mathcal{POI}_4[1,20])$	16	4	1807	40	491	30
$Min(\mathcal{POI}_5[1,125])$	32	5	107438	270	8602	130
$Min(\mathcal{POPI}_4[1,60])$	33	2	8381	40	704	30
$Min(\mathcal{POPI}_5[1,70])$	81	2	398620	340	11528	260

3.2 Functions to Compute "Simpler" Equivalent Automata

Many applications of, and computations with, finite state automata naturally produce non-deterministic automata with ϵ-transitions. On the other hand, many algorithms require a deterministic automaton as input. There are standard algorithms in the literature which address this question, first converting a non-deterministic finite state automaton with ϵ-transitions to an equivalent one without ϵ-transitions, and then to a (possibly exponentially larger) equivalent deterministic automaton. There is also an essentially unique minimal deterministic automaton equivalent to any given automaton, and again standard algorithms exist for determining it.

The package contains highly efficient implementations of these algorithms, optimized for handling automata with many (possibly millions) of states.

In addition to this "standard" functionality, two further functions are provided in this area. The first takes a non-deterministric automaton with ϵ-transitions and identifies states which are mutually reachable by ϵ-transitions alone. This function is called `EpsilonCompactedAut` and, thanks to an efficient implementation of a standard strongly connected components algorithm for directed graphs, is extremely quick, making it a useful pre-processing step before converting to an automataton without ϵ-transitions. The second function `ReducedNFA` implements the algorithm in [AL04] for computing a smaller non-deterministic automaton equivalent to a given one, which is often a useful pre-processing step before determinizing.

4 A package for semigroups

The GAP deposited package *sgpviz* [DM05b], which aims to turn GAP more user-friendly for finite semigroup theorists, has strong connections with the package *automata*. For example, automata intervenes when one wants to

specify a finite semigroup as the syntactic semigroup of a rational language. This is just one of the ways a semigroup can be easily done to GAP using *sgpviz*.

The package *sgpviz* requires the usage of external programs as is the case of *graphviz* [DEG+02], a software for drawing graphs developed at AT & T Labs. Recall that this software is also used by *automata* if one wants to visualize the drawing of an automaton. Here *graphviz* is used not only to draw right Cayley graphs of finite semigroups and Schüzenberger graphs of finite inverse semigroups but also to visualize in the usual way the egg-box picture of a D-class of a finite semigroup. Of course, most of these operations use functions that are part of GAP.

It requires also the use of Tcl/Tk. The package contains Tcl/Tk programs to help the user to specify automata and semigroups.

References

[AL04] M. H. Albert and S. Linton. A practical algorithm for reducing non-deterministic finite state automata. Technical report, Otago university CS tech report OUCS 2004-11, 2004. (`http://www.cs.otago.ac.nz/research/techreports.html`).

[DEG+02] D. Dobkin, J. Ellson, E. Gansner, E. Koutsofios, S. North, and G. Woodhull. Graphviz - graph drawing programs. Technical report, AT&T Research and Lucent Bell Labs, 2002. (`http://www.graphviz.org/`).

[Del98] M. Delgado. Abelian pointlikes of a monoid. *Semigroup Forum*, 56:127–146, 1998.

[Del01] M. Delgado. Commutative images of rational languages and the abelian kernel of a monoid. *Theoretical Informatics and Applications*, 35:419–435, 2001.

[DLM05] M. Delgado, S. Linton, and J. Morais. automata*: a GAP package for finite automata, Version 1.05*, 2005. (`http://www.gap-system.org/Packages/automata.html`).

[DM05a] M. Delgado and J. Morais. Approximation to the smallest regular expression for a given regular language. In K. Salomaa

M Domaratzki, A. Okhotin and S. Yu, editors, *Implementation and Application of Automata*, pages 312–314, Heidelberg, 2005. Springer-Verlag.

[DM05b] M. Delgado and J. Morais. sgpviz*: a GAP package to visualize finite semigroups, Version 0.991*, 2005. (`http://www.gap-system.org/Packages/sgpviz.html`).

[GAP04] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.4*, 2004. (`http://www.gap-system.org`).

[KM02] I. Kapovich and A. Myasnikov. Stallings foldings and subgroups of free groups. *J. of Algebra*, 248:608–668, 2002.

[MMP+95] O. Matz, A. Miller, A. Potthoff, W. Thomas, and E. Valkema. Report on the program amore. Technical Report 9507, Christian Albrechts Universität, Kiel, 1995.