COMPUTING WITH SEMIGROUPS AND MONOIDS

Steve Linton

Division of Computer Science, St. Andrews

July 7, 1997

Overview

1. Transformation Monoids
   - with Götz Pfeiffer, Ed Robertson, Nik Ruškuc
   - Computer investigations
   - Theoretical results inspired by investigations
   - Improved algorithms justified by new theory
2. Computation in Algebra
   - Roles of Computation
   - Capabilities of Modern Software
   - Theoretical questions arising

TRANSFORMATION MONOIDS

The Basic Problem

Input: A finite set of generating transformations A given by images i^a for $1 \leq i \leq n$ and $a \in A$.

Output: Some or all of:

- $|M| = |\langle A \rangle|$
- $\mathcal{D}$, $\mathcal{L}$ and $\mathcal{R}$ class representatives, a parameterization of the $\mathcal{H}$ classes
- Fast membership test for M
- Fast tests for $\mathcal{D}$, $\mathcal{L}$, $\mathcal{R}$ or $\mathcal{H}$

Previous Work

- Basic algorithm: lists all the elements of M, standard algorithms for directed graphs will find the Green's classes
- Lallement-McFadden: work down $\mathcal{D}$ classes.
  - regular classes: represent a group $\mathcal{H}$ class as a permutation group on few points, this controls the rest of the $\mathcal{D}$ class
  - non-regular classes: some control is exerted by $\mathcal{D}$ classes above, but the algorithms still list all elements for some purposes

Our First Experiments

- We re-implemented L-McF in GAP
- A lot easier than their original implementation – lists and permutations built in
- We found many suspiciously composite numbers – of $\mathcal{L}$ and $\mathcal{R}$ classes in a $\mathcal{D}$ class, for example.
- This suggested that
  - The groups that control the non-regular $\mathcal{D}$ classes might also be small degree permutation groups
  - There might be larger groups controlling the regular $\mathcal{D}$ classes than we knew about
- We arrived eventually at the theory in our paper "Groups and actions in transformation semigroups" to appear in Math Zeitschrift

Actions and Orbits

- Let a monoid M act on a set Ω such that
$$(\omega m)m' = \omega(mm')$$
- Define the *cone* $C(\omega)$ of $\omega \in \Omega$ to be $\{\omega m : m \in M\}$
- Define the *strong orbit* $O(\omega)$ of $\omega \in \Omega$ to be $\{\omega m : m \in M, \omega \in C(\omega m)\}$
- Strong orbits are equivalence classes
- Note that an $\mathcal{R}$ class is a strong orbit in the regular action
- Define an action on subsets of Ω by $\Delta m = \{\delta m : \delta \in \Delta\}$
- Define the stabilizer $\mathrm{Stab}_M(\Delta) = \{m \in M : \Delta m = \Delta\}$ (*not* $\Delta m \subseteq \Delta$)
- Observe that $G(\Delta) = \mathrm{Stab}_M(\Delta)|_\Delta$ is a group of permutations of Δ.
- This is called the “Generalised Schützenberger Group of Δ”

Computing Generalized Schützenberger Groups

Theorem 1 *Let $M = \langle A \rangle$ be a monoid acting on a set Ω. Let $\Delta \leq \Omega$ and let*
$$O(\Delta) = \{\Delta = \Delta_1, \Delta_2, \ldots, \Delta_m\}.$$
Let $r_1, \ldots, r_m$ and $r'_1, \ldots, r'_m$ be elements of M such that $\Delta r_i = \Delta_i$ and $\Delta_i r'_i = \Delta$. Then
$$G(\Delta) = \langle (r_i a r'_k)|_\Delta : a \in A, \Delta_i a = \Delta_k \rangle$$

- This is a version of the Schreier Generator Theorem from group theory
- It is easy to modify the strongly connected components algorithm to compute, simultaneously:
  - $O(\Delta)$
  - $r_1, \ldots, r_k$ and $r'_1, \ldots, r'_k$
  - the generators of $G(\Delta)$

$\mathcal{R}$ Classes

- For t a transformation, we call $G(\operatorname{Im} t)$ the "right Generalised Schützenberger Group of t" and write $G_R(t)$
- It is easy to show that:

$$\{\operatorname{Im} s : s \,\mathcal{R}\, t\} = O(\operatorname{Im} t)$$

- Furthermore, $\{s : s \,\mathcal{R}\, t, \operatorname{Im} s = \operatorname{Im} t\}$ is in natural bijection with $G_R(t)$
- In fact, the whole $\mathcal{R}$-class $t\mathcal{R}$ can be placed in bijection with $O(\operatorname{Im} t) \times G_R(t)$
- This bijection is computationally nice. After doing the computation of $G_R(t)$ above, we have a constructive numbering of $t\mathcal{R}$ and so know:
  - $|t\mathcal{R}|$
  - how to test an element for membership of $t\mathcal{R}$
  - how to run through the elements of $t\mathcal{R}$

  These depend on fast algorithms for permutation groups

Studying Whole Monoids

- $\mathcal{R}$ classes are a left congruence
- We can find them all quite efficiently as the "left cone" of $1\mathcal{R}$
- If $\operatorname{Im} s = \operatorname{Im} t$ then $G_R(s) = G_R(t')$ and $O(\operatorname{Im} s) = O(\operatorname{Im} t)$, so $s\mathcal{R}$ and $t\mathcal{R}$ can be described by almost the same data structure, which need only be constructed once
- This is the method used by the `Size` function in our package

Duality, $\mathcal{L}$ classes, $\mathcal{D}$ classes

- We can also define and compute the "left generalised Schützenberger group" $G_L(s)$ by working with kernels instead of images
- This controls $\mathcal{L}$-class structure
- Further we can view $G_L(s)$ as a permutation group on $\mathrm{Im}\, s$
- $G_L(s) \cap G_R(s)$ is isomorphic to the Schützenberger group $\Gamma(s\mathcal{H})$
- The whole of $t\mathcal{D}$ has a rectangular structure:
  - each block is in bijection to the group product $G_L(t)G_R(t)$
  - the blocks are parameterized by

$$\{\mathrm{Im}(s) : s\mathcal{R}t\} \times \{\mathrm{Ker}(s) : s\mathcal{L}t\}$$

Computing with Semigroups Transformation Monoids

The Monoid Package

- `Monoid` is a contributed package extending GAP 3
- Basic data structures and functions for semigroups, monoids, Green classes, etc.
- Data structures for transformations and binary relations
- Algorithms for transformation monoids, and other monoids acting on sets, based on new theory

COMPUTATIONAL ALGEBRA IN GENERAL

History – Computational Group Theory

- Algorithmic methods in group theory date back to the 1930s
- Electronic computers were used in the 1950s
- Computational results played a part in the Classification of Finite Simple Groups, and in the subsequent analysis of the sporadic groups
- Since the 1970s, integrated systems have been developed for areas, and then for the whole, of group theory
- They provide a way to store groups, elements and other structures, a range of standard algorithms to work with them, and a programming language for combining these
- In the 1990s, more sophisticated algorithms have been developed, relying on sophisticated mathematics and on much prior art. This has:
  - pushed development of integrated systems – without such a system many new algorithms cannot be implemented within a single PhD
  - forced the systems to broaden beyond group theory – polynomials, algebras, ...

The Reds and the Greens

- Two schools in CGT (especially pre 1991)
- Green school (Sims, Europeans, Australians, mainly mathematicians)
  - Pragmatists
  - Implementation is important
  - "Good" algorithm performs well on standard or big examples
- Red school (North America, mainly computer scientists)
  - Complexity theorists – started from Graph Isomorphism problem
  - Precise complexity calculation important
  - "Good" algorithm has low asymptotic complexity
- Some fusion since 1991 – Seress, Cooperman
- Result – better theory and better implementations

Random Algorithms

- Some algorithms make random choices as they run
- Las Vegas algorithms may (if unlucky) fail, but never give a wrong answer
- Monte Carlo algorithms give the wrong answer with some probability $\epsilon < 1/2$. Repeated runs will reduce ϵ as required
- To make a MC algorithm LV simply need a check on the final answer

What can be done

- Finitely-presented groups:
  - Coset enumeration up to $c\ 10^6$ cosets
  - Study quotients: abelian, p, finite, nilpotent, polycyclic
  - Seek confluent rewriting systems or automatic structures
- Permutation groups:
  - Most structural questions can be answered for reasonable groups up to degree 10^6 (sometimes Monte Carlo)
  - Search problems within the groups can be harder
- Finite Soluble groups:
  - Almost any question can be answered for quite large groups
  - Work up and down the group and reduce problems to linear algebra

Integrated Systems

- GAP, MAGMA. Both provide:
  - implementations of a wide range of algorithms
  - "glue" to allow algebraic data to be passed between them
  - support features like memory management, large integer arithmetic, finite field arithmetic, lists, records, etc.
  - a programming language with which the user can extend the system, drawing freely on all the existing features

Integrated Systems 2

- Why use such a system?
  - The techniques you need may be built in – use as a "desk calculator" – no user programming at all
  - Interpreted language and interactive debugging make it easier to program in GAP or MAGMA than in (say) C, especially small programs
  - Many modern algorithms draw on a huge range of known techniques – avoid the need to reimplement them all
  - Existing system capabilities are there to prepare input and analyse output from your new algorithm
  - Easy for others to build on your work
- Why not?
  - Performance penalty – 3 to 10 times for most things

GAP

- Groups, Algorithms, Programming
- Free software
- Runs under UNIX, MSDOS and MacOS (at least) – easy to port
- Developed over 11 years in Lehrstuhl D für Mathematik, RWTH Aachen, Germany
  - Prof Joachim Neubüser
  - Martin Schönert
  - Cast of thousands, mainly in Aachen, some elsewhere
- Headquarters moving to St Andrews in the next few weeks
- Current version 3.4.4. Version 4 about to enter β-test
- System is mostly written in the GAP language

GAP Contd

- Strongest areas
  - Permutation groups (esp in v4)
  - Finite Soluble groups
  - Character tables
  - Programmability
  - A number of powerful contributed share packages
- More info `http://www-gap.dcs.st-and.ac.uk/~gap/`

MAGMA

- Not free ($1000 – $2000 for a 3 year license?)
- Successor to Cayley
- Produced by John Cannon in Sydney
- Available for UNIX, DOS, maybe MacOS
- Current version 2.2
- Strong in number theory, representation theory, finitely-presented groups
- Mostly written in C – incorporated many stand-alone C programs
- High performance within the C modules
- Less popular with programmers and extenders

Theory from Computation

- As well as suggesting answers, computation has led to interesting new questions
- To prove (worst-case) complexity results requires "extremal" group theory – eg if it is not one of these special cases, how large can a permutation group of given degree be
- To prove probabilities for random algorithms requires statistical information about groups – for example finding provably random elements of a group from the generators is surprisingly hard

Diversification

- Group theoretic algorithms are now far more sophisticated than those for other structures
- Can we apply the same ideas, or at least some of the general lessons in other contexts?
- Recent progress with
  - Lie Algebra (Willem de Graaf)
  - Finitely-presented semigroups (Robertson, Thomas, Ruskuc, *et al*)
  - Transformation Semigroups (SL, Pfeiffer, Ruskuc, Robertson)
  - Finitely-presented associative algebras (SL)
- Interest in
  - Matrix rings and algebras
  - Groupoids
  - Quantum groups
- So far, examples support the general thesis
- GAP 4 is designed to support this work much better than GAP 3