JSAG **5** (2013), 15 – 19

The Journal of Software for
Algebra and Geometry

Computing the invariant ring of a finite group

THOMAS HAWES

ABSTRACT. We give an overview of a new package for *Macaulay2* called *InvariantRing*, which contains tools for describing the invariant ring of finite group actions on polynomial rings in characteristic zero. We outline methods for computing primary and secondary invariants and compare the two algorithms that are implemented for computing primary invariants.

INTRODUCTION. Let k be a field of characteristic zero and $\mathrm{GL}(n,k)$ the group of invertible $n \times n$ matrices. Let $S := k[x_1,\dots,x_n]$ be the ring of polynomial functions on k^n, where elements of k^n are considered as column vectors. We write $\mathbf{x} = (x_1,\dots,x_n)^{\mathrm{t}}$ for the column vector of the variables of S and let elements of $\mathrm{GL}(n,k)$ act on $\mathbf{x}$ by matrix multiplication. For a finite subgroup $G \leq \mathrm{GL}(n,k)$, consider the left action on S defined by $(A \cdot f)(\mathbf{x}) := f(A^{-1}\mathbf{x})$, for any $A \in G$ and $f \in S$. The ***invariant ring*** of the group G is the ring $S^G := \{f \in S \,|\, A \cdot f = f \text{ for all } A \in G\}$ and its elements are called ***invariants***. Due to the fact that G acts on S by linear transformations, a polynomial is invariant if, and only if, each of its homogeneous pieces is invariant; thus S^G is a graded subring of S. The following structure theorem, which can be found in [N, DK], provides the main motivation for the package *InvariantRing*.

Theorem 1. *For any finite group $G \leq \mathrm{GL}(n,k)$, there exist homogeneous, algebraically independent invariants $f_1,\dots,f_n$ and homogeneous invariants $g_1,\dots,g_r$ such that*

$$(\ddagger) \qquad S^G = \bigoplus_{j=1}^{r} k[f_1,\dots,f_n]\, g_j.$$

The $f_1,\dots,f_n$ are called ***primary invariants*** and the $g_1,\dots,g_r$ ***secondary invariants***. The purpose of the package *InvariantRing* for *Macaulay2* [M2] is to compute primary and secondary invariants for the invariant ring of a finite group over number fields.

Collections of primary and secondary invariants for an invariant ring are not uniquely determined, but once a system of primary invariants has been chosen, the degrees of any corresponding system of secondary invariants are determined by the following theorem (see [DK]):

Theorem 2. *Let $f_1,\dots,f_n$ be primary invariants for S^G with degrees $d_1,\dots,d_n$, and let $H(S^G,t) \in \mathbb{Z}[t]$ be the Hilbert series of S^G. Suppose there are r corresponding secondary invariants.*

(a) *$H(S^G,t) \cdot \prod_{i=1}^{n}(1-t^{d_i}) = t^{e_1} + \cdots + t^{e_r}$, where $e_1,\dots,e_r$ are the degrees of any corresponding secondary invariants. In particular, this polynomial has integer coefficients.*

(b) *The number of secondary invariants is $r = \frac{d_1 \cdots d_n}{|G|}$. In particular, $|G|$ divides $d_1 \cdots d_n$.*

2010 *Mathematics Subject Classification.* 13-04.

InvariantRing version 1.1.0.

It is easy to compute the Hilbert series of S^G by using Molien's theorem, which states that $H(S^G,t) = \frac{1}{|G|}\sum_{A\in G}\det(I_n - tA)^{-1}$, where I_n is the $n\times n$ identity matrix [N]. Molien's theorem, along with Theorem 2, gives the number of secondary invariants and their degrees.

The package *InvariantRing* provides tools to study and compute the invariant ring of a finite group action on a polynomial ring in characteristic zero. In the next section, we describe the key features of the package and discuss the methods available for computing decompositions of the invariant ring as in (‡). We conclude the article with a demonstration of these features by way of an extended example.

OVERVIEW. The package *InvariantRing* contains the methods `generateGroup`, `reynoldsOperator`, and `molienSeries` as tools for studying the invariant ring of a finite group. To aid the input of finite groups of matrices, the method `generateGroup` takes a list of generating matrices $\{A_1,\dots,A_m\}$ and outputs the group they generate, using a brute-force algorithm that computes all possible products of the A_i. The method `reynoldsOperator` uses a stored group G to compute the average $\frac{1}{|G|}\sum_{A\in G} A\cdot f$ of a polynomial $f\in S$. Finally, the method `molienSeries` computes the Hilbert series $H(S^G,t)$ of S^G using Molien's theorem, expressing the result as a rational expression of the `divide` class.

The core of the package consists of methods for computing a decomposition as in (‡). The method `primaryInvariants` implements two different algorithms for computing primary invariants for S^G: the 'optimal' algorithm from [K2] and the 'Dade' algorithm, described in [DK, S], for example. The first algorithm is the default for the method `primaryInvariants`. It begins by cycling through n-tuples $(d_1,\dots,d_n)\in\mathbb{N}^n$, ordered by increasing values of the product $d_1\cdots d_n$. For each tuple, it tests whether primary invariants with degrees $d_1,\dots,d_n$ can exist, firstly by using the two conditions from Theorem 2 and then by using the Krull dimension test from [K2, Theorem 2(b)]. If these tests are passed, a set of primary invariants with the proposed degrees is constructed iteratively using the same dimension test; otherwise, the next degree vector is considered. The resulting primary invariants have coefficients that are integers or uncomplicated fractions. In addition, the product of their degrees is as small as possible, so the number and degrees of secondary invariants needed in (‡) are kept small, by Theorem 2.

The default routine in the method `primaryInvariants` cannot compute primary invariants when working over ground fields of positive characteristic, even if the characteristic is coprime to $|G|$. In this case, the method `primaryInvariants` can be used with the option `Dade` set to `true` to find a system of primary invariants using the Dade algorithm instead, provided the cardinality of the ground field is sufficiently large. The algorithm works by finding a 'Dade basis' $v_1,\dots,v_n$ of S_1, then computing $f_i = \prod\{w \mid w\in \mathrm{Orb}_G(v_i)\}$ for each i; the collection $f_1,\dots,f_n$ is then a set of primary invariants. The construction of a Dade basis involves only choosing random linear forms and linear algebra, so has the advantage of being simple and quick. However, the resulting primary invariants almost always have ugly coefficients and have degrees being the same as the order of the group. When over a characteristic zero field with a small group, the Dade algorithm is often faster than the default algorithm, but at the cost of calculating more secondary invariants to obtain a decomposition as in (‡).

The method `secondaryInvariants` uses the primary invariants $f_1,\dots,f_n$ and their degrees $d_1,\dots,d_n$ to compute corresponding secondary invariants, by finding a collection of $r = d_1\cdots d_n/|G|$ homogeneous invariants whose degrees are the values predicted in Theorem 2(a) and whose images under the projection $S\to S/(f_1,\dots,f_n)$ are linearly independent over k (for a justification, see [DK, §3.5.1]).

Thus, finding secondary invariants boils down to a question of doing linear algebra with normal forms of polynomials with respect to the ideal $(f_1,\dots,f_n) \subset S$. The implementation of this procedure in `secondaryInvariants` only works when the ground field is of characteristic zero.

The package also includes the method `invariantRing` for computing a decomposition as in (‡) directly, based only on the data of the polynomial ring S and the group G. It outputs a collection of primary invariants, computed using Kemper's 'optimal' algorithm from [K2], and a corresponding collection of secondary invariants.

As indicated above, the package *InvariantRing* cannot compute the full invariant ring of a finite group over ground fields of positive characteristic. However, there are algorithms in the literature that could be implemented in the future to address this. In particular, [K1, K2] give algorithms for computing primary and secondary invariants of S^G when $\gcd(|G|, \operatorname{char} k) = 1$. When $\operatorname{char} k$ divides $|G|$, the ring S^G is in general not Cohen-Macaulay [K1, Example 13], so does not admit a decomposition as in (‡). In this case, the algorithms in [K1, K2] compute homogeneous, algebraically independent invariants $f_1,\dots,f_n$, together with homogeneous invariants $g_1,\dots,g_r$ generating S^G as a (not necessarily free) $k[f_1,\dots,f_n]$-module.

AN EXAMPLE. We conclude this article with an example that demonstrates the main methods available in *InvariantRing* and compares the default and Dade algorithms for calculating primary invariants. The group G considered is the dihedral group of order 8, generated by matrices

$$A = \begin{bmatrix} -1 & 1 & 0 \\ -1 & 0 & 1 \\ -1 & 0 & 0 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 & -1 & 1 \\ -1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}.$$

This group acts on the polynomial ring $S = \mathbb{Q}[x,y,z]$.

To begin, the print width is altered, the package *InvariantRing* is loaded, and the matrices and polynomial ring input.

```
Macaulay2, version 1.6
with packages: ConwayPolynomials, Elimination, IntegralClosure, LLLBases,
               PrimaryDecomposition, ReesAlgebra, TangentCone
i1 : printWidth = 70; truncateOutput 140; needsPackage "InvariantRing";
i4 : A = matrix{{-1,1,0},{-1,0,1},{-1,0,0}}; B = matrix{{0,-1,1},{-1,0,1},{0,0,1}};
             3       3
o4 : Matrix ZZ  <--- ZZ
             3       3
o5 : Matrix ZZ  <--- ZZ
i6 : S = QQ[x,y,z];
```

We generate a group G from the list of matrices `{A,B}` using the `generateGroup` method. The field in the second argument ensures the resulting matrices are defined over $\mathbb{Q}$, the ground field of S.

```
i7 : G = generateGroup({A,B},QQ)
o7 = {| 0 -1 1 |, | 0  0  -1 |, | -1 1 0  |, | 0 0 -1 |, | 1 0  0  |,
      | 0 -1 0 |  | 0  -1 0  |  | 0  1 0  |  | 1 0 -1 |  | 1 0  -1 |
      | 1 -1 0 |  | -1 0  0  |  | 0  1 -1 |  | 0 1 -1 |  | 1 -1 0  |
     ------------------------------------------------------------------
     | 0  -1 1 |, | -1 1 0 |, | 1 0 0 |}
     | -1 0  1 |  | -1 0 1 |  | 0 1 0 |
     | 0  0  1 |  | -1 0 0 |  | 0 0 1 |
o7 : List
```

Next, we find a list of primary invariants for S^G using `primaryInvariants`. We do this using both the default algorithm and the Dade algorithm, the latter being called by setting the optional argument `Dade` to `true`. We obtain the degrees of the primary invariants computed with the Dade algorithm, but truncate the full output.

```
i8 : time prim1=primaryInvariants(S,G)
     -- used 0.534851 seconds

       2    2           2   2          2          2   4    4     3
o8 = {x  + y  - 2x*z + z , x  - x*y + y  - y*z + z , x  + y  - 4x z +
     ------------------------------------------------------------------
       2 2       3    4
     6x z  - 4x*z  + z }

o8 : List

i9 : time prim2=primaryInvariants(S,G,Dade=>true)
     -- used 0.37052 seconds

o9 =             8              7                6 2              5 3
     {1536640000x  - 5976432000x y - 53745983200x y  + 9626324400x y
     ------------------------------------------------------------------
                    4 4              3 5               2 6
     + 128235861600x y  + 9626324400x y  - 53745983200x y  -
     ------------------------------------------------------------------
                  7            ...
     5976432000x*y  + 15366400 ...

o9 : List

i10 : apply(prim2,degree)

o10 = {{8}, {8}, {8}}

o10 : List
```

The Dade algorithm was faster than the default algorithm in this example, but the resulting invariants are complicated polynomials of degrees equalling the order of the group, which will influence the time needed to compute secondary invariants. This is done next by applying the method `secondaryInvariants` to the lists `prim1` and `prim2`. According to Theorem 2(b) there are $8^3/8 = 64$ secondary invariants corresponding to the collection `prim2`, so again we suppress the output.

```
i11 : time sec1=secondaryInvariants(prim1,G)
     -- used 0.026354 seconds

           2       2    2       2
o11 = {1, x y - x*y  + y z - y*z }

o11 : List

i12 : time sec2=secondaryInvariants(prim2,G);
     -- used 3.83254 seconds
```

We note how much quicker it was to compute the secondary invariants for `prim1` than `prim2`. In fact, on this occasion the default algorithm for computing primary invariants resulted in a quicker overall computation of the decomposition (‡) for the invariant ring of G.

To see how many corresponding secondary invariants of each degree there are, we compute the Hilbert series of S^G using the method `molienSeries` and Theorem 2(a).

```
i13 : mol = molienSeries G
```

```
                          2
              - 1 + T - T
o13 = -------------------------
                3      2        2
      (- 1 + T) (1 + T )(1 + T)
o13 : Expression of class Divide
i14 : T = first gens ring numerator mol;
i15 : ((value numerator mol)*(1-T^8)^3)//(value denominator mol)
             2    3     4     5     6     7     8     9     10     11
o15 = 1 + 2T  + T  + 4T  + 2T  + 6T  + 4T  + 6T  + 6T  + 6T   + 6T   +
      ------------------------------------------------------------------
        12     13     14     15    16     17    19
      4T   + 6T   + 2T   + 4T   + T   + 2T   + T
o15 : ZZ[T]
```

Thus, for example, we see that there are 6 secondary invariants of degree 11. To print this polynomial during the construction of secondary invariants, we call `secondaryInvariants` with the optional argument `PrintDegreePolynomial` set to `true`:

```
i16 : secondaryInvariants(prim2,G,PrintDegreePolynomial=>true);
 19     17    16     15     14     13     12     11     10     9     8     7     6 ...
t   + 2t   + t   + 4t   + 2t   + 6t   + 4t   + 6t   + 6t   + 6t  + 6t  + 4t  + 6t  ...
```

Finally, we mention that the method `invariantRing` computes primary and secondary invariants in one go, calling upon the method `primaryInvariants` with the optional argument `Dade` set to `false`. Executing the command `invariantRing(S,G)` in the above example would output `{prim1,sec1}`.

Acknowledgement. The author wishes to express his gratitude to Diane Maclagan for suggesting and supervising the writing of this package. He also thanks an anonymous referee and Amelia Taylor for helpful comments concerning the code and this article.

REFERENCES.

[DK] H. Derksen and G. Kemper, *Computational invariant theory*, Invariant Theory and Algebraic Transformation Groups, I, Springer-Verlag, Berlin, 2002.

[K1] G. Kemper, *Calculating invariant rings of finite groups over arbitrary fields*, J. Symbolic Comput. **21** (1996), no. 3, 351–366.

[K2] ———, *An algorithm to calculate optimal homogeneous systems of parameters*, J. Symbolic Comput. **27** (1999), no. 2, 171–184.

[M2] D.R. Grayson and M.E. Stillman, *Macaulay2, a software system for research in algebraic geometry*, available at www.math.uiuc.edu/Macaulay2/.

[N] M.D. Neusel, *Invariant theory*, Student Mathematical Library, vol. 36, American Mathematical Society, Providence, RI, 2007.

[S] B. Sturmfels, *Algorithms in invariant theory*, 2nd ed., Texts and Monographs in Symbolic Computation, SpringerWienNewYork, Vienna, 2008.

RECEIVED : 2012-08-10 REVISED : 2013-03-24 ACCEPTED : 2013-05-16

thomas.hawes@maths.ox.ac.uk : Balliol College, Broad Street, Oxford, OX1 3BJ, UK.