J. Symbolic Computation (1995) **20**, 755–768

A Maple Package for Symmetric Functions

JOHN R. STEMBRIDGE[†‡]

Dep. of Mathematics, University of Michigan, Ann Arbor, Michigan 48109-1003

(Received 1 November 1993)

We describe the main features of a package of Maple programs for manipulating symmetric polynomials and related structures. Among the highlights of the package are (1) a collection of procedures for converting between polynomial expressions involving several fundamental bases, and (2) a general mechanism that allows the user to easily add new bases to the existing collection. The latter facilitates computations involving numerous important families of symmetric functions, including Schur functions, Zonal polynomials, Jack symmetric functions, Hall-Littlewood functions, and the two-parameter symmetric functions of Macdonald.

1. Introduction

The purpose of this article is to announce the release of version 2 of SF, a free package of 23 *Maple* programs that provide an environment for computations involving symmetric functions and related structures. This package is likely to be (and already has been) useful for computations related to

- Polya-Redfield-style enumeration and tableau enumeration.
- Characters of the symmetric groups and the (complex) general linear groups.
- Classical invariant theory.
- Hall-Littlewood functions and Kostka-Foulkes polynomials.
- Zonal spherical functions for GL_n/O_n.
- Macdonald's two-parameter symmetric functions.

There are numerous improvements that have been incorporated into this second version of SF, including:

† *Partially supported by NSF grants DMS–9057192 and DMS–9101898*
‡ E-mail: `jrs@math.lsa.umich.edu`

0747–7171/95/050755+14 $12.00/0

- Added functionality, such as the ability to dynamically add new symmetric function bases during a Maple session.
- Many of the procedures have been rewritten and are now significantly faster and more space-efficient.
- Improved documentation.
- Help texts for the individual commands are now incorporated into the Maple help facility.

1.1. System Requirements

In principle, there is no reason that SF cannot be run on any system that has *Maple V* (Release 1 or 2). However, the package in its current form takes advantage of the UNIX file structure, and hence those who wish to use the package with another operating system will have to duplicate the arrangement of files and directories used by SF. Given the file handling facilities provided by Maple itself, this should not be too difficult.

The complete package includes the source code, help texts, a test file (for verifying installation), a 15 page manual, and an "examples" directory consisting of additional programs that provide applications of, or extensions to, SF. The compiled code occupies about 50K bytes of disk space, and the entire package, after installation, occupies about 250K bytes of disk space.

1.2. Availability

Eventually, this package should be added to the Maple Share Library. When that happens, it should be possible to run this package on any hardware that runs Maple. However until this occurs, one will need to have access to a UNIX-based host on the Internet to acquire and unpack SF. Currently, the package is available by anonymous ftp from the host `ftp.math.lsa.umich.edu` in the directory `pub/jrs`. More detailed instructions can be obtained by sending an e-mail message to the author.

The remainder of this article is concerned with giving an overview of the package, explaining the types of data structures (§3) and algorithms (§4) used, and summarizing some fundamental parts of the mathematical theory (§2). A short synopsis of the purpose of each procedure in the package is provided in §5.

2. Some definitions

First we briefly review some of the basic definitions in the theory of symmetric functions. It is highly recommended, but not strictly necessary, that the reader be previously acquainted with the definitive treatment by Macdonald (1979). Alternative sources are

Fulton and Harris (1991) (Appendix A), Littlewood (1950), Sagan (1991), and Stanley (1971), although some of these use notation that does not conform to Macdonald's.

By a symmetric polynomial, we mean a polynomial $f = f(x_1, x_2, \ldots, x_n)$ that is invariant under permutations of $x_1, \ldots, x_n$. The symmetric polynomials of course form a graded subring of the ring of polynomials functions of $x_1, \ldots, x_n$. By specialization (setting some of the $x_i = 0$), one finds that for most purposes, the number of variables in a symmetric polynomial is irrelevant, as long as the number is sufficiently large. Taking this idea to its logical extreme, one is led to consider symmetric "polynomials" in infinitely many variables†, say $x_1, x_2, \ldots$. These symmetric "polynomials," also known as symmetric functions, form a graded ring Λ. For a more rigorous definition, see Macdonald (1979).

There are several families of symmetric functions that are of fundamental importance; namely, the elementary symmetric functions $e_1, e_2, \ldots$, defined by

$$e_r = \sum_{1 \le i_1 < i_2 < \cdots < i_r} x_{i_1} \cdots x_{i_r},$$

the complete homogeneous symmetric functions $h_1, h_2, \ldots$, defined by

$$h_r = \sum_{1 \le i_1 \le i_2 \le \cdots \le i_r} x_{i_1} \cdots x_{i_r},$$

and the power-sum symmetric functions $p_1, p_2, \ldots$, defined by

$$p_r = \sum_{i \ge 1} x_i^r.$$

The most basic result in the theory of symmetric functions is the fact that the elementary symmetric functions $e_1, e_2, \ldots$ are algebraically independent (over $\mathbf{Q}$, say) and generate the ring Λ. Consequently, the set of all monomials in the e_r's form a vector space basis of Λ. Since e_r is homogeneous of degree r, it follows that the dimension of the n*th graded component of* Λ *is* $p(n)$, *the number of partitions of* n *into an (unordered)* sum of positive integers. If $n = \lambda_1 + \cdots + \lambda_l$ is such a partition, it is convenient to define

$$e_\lambda := e_{\lambda_1} \cdots e_{\lambda_l},$$

so that as λ varies over all partitions, the e_λ's yield a basis for Λ.

All of the above remarks apply equally well to the h_r's and the p_r's. In particular, the h_λ's and p_λ's each form bases for Λ.

There are two additional fundamental bases of Λ, the monomial symmetric functions m_λ and the Schur functions s_λ. In the former case, assuming $\lambda = (\lambda_1, \ldots, \lambda_l)$, we have

$$m_\lambda = \sum x_{i_1}^{\lambda_1} \cdots x_{i_l}^{\lambda_l},$$

† Strictly speaking, these are no longer polynomials, but formal power series.

where the sum ranges over all *distinct* monomials whose exponent sequence is some permutation of λ. For the Schur functions, the briefest (and least motivated) definition is

$$s_\lambda = \det[h_{\lambda_i - i+j}]_{1 \le i,j \le l},$$

with the conventions $h_0 = 1$ and $h_{-r} = 0$ for $r > 0$. The importance of Schur functions derives from their connection with the irreducible characters of the symmetric groups and general linear groups. (For more about this, see Fulton and Harris (1991), James and Kerber (1981), or Macdonald (1979).)

There is one obvious distinction between the first three bases we defined (e_λ, h_λ, and p_λ) and the latter two (m_λ and s_λ). The latter are not multiplicative. In other words, only in the former cases do we have a basis b_λ such that $b_\lambda = b_{\lambda_1} b_{\lambda_2} \cdots$ whenever $\lambda = (\lambda_1, \lambda_2, \ldots)$.

There is a convenient scalar product on Λ that can be defined (and is over-determined) by the properties

$$\langle h_\lambda, m_\mu \rangle = z_\lambda^{-1} \langle p_\lambda, p_\mu \rangle = \langle s_\lambda, s_\mu \rangle = \delta_{\lambda\mu}, \tag{2.1}$$

where $z_\lambda := \prod_i a_i! \, i^{a_i}$, assuming there are a_i occurrences of i as a term of λ. In connection with this combined definition-assertion it is significant that the transition matrix between the Schur functions and monomial symmetric functions is triangular. In fact,

$$s_\lambda = \sum_{\mu \le \lambda} K_{\lambda,\mu} m_\mu \tag{2.2}$$

for suitable integers $K_{\lambda,\mu}$ (the Kostka numbers). Here the notation '$\mu \le \lambda$' refers to any total ordering of the set of partitions that refines the dominance partial order. (This is the partial order obtained by imposing the inequalities $\mu_1 + \cdots + \mu_i \le \lambda_1 + \cdots + \lambda_i$ for all $i \ge 1$.) A consequence of (2.1) and (2.2) is that the Schur functions are the unique orthonormal basis of Λ (relative to $\langle \, , \, \rangle$) that one would obtain by applying the Gram–Schmidt algorithm to the m-basis. Alternatively, using the fact that $K_{\lambda,\lambda} = 1$, one can characterize the Schur functions as the unique orthogonal basis of Λ satisfying

$$s_\lambda = m_\lambda + \text{ a linear combination of } m_\mu \text{ for } \mu < \lambda.$$

This point of view provides a unified method for treating many other important families of symmetric functions. To be more precise, there are a number of bases in the literature on symmetric functions that can be characterized (up to scalar multiples) by the properties of having a triangular transition matrix with respect to the monomial symmetric functions and being orthogonal with respect to some scalar product.

The following is a list of examples of such bases. Note that in every case, the power sums p_λ are orthogonal with respect to the relevant scalar product.

1. The Hall-Littlewood functions $HL_\lambda(t)$ [see Macdonald (1979)]. These are orthogonal

relative to the scalar product

$$\langle p_\lambda, p_\mu \rangle_t := \delta_{\lambda\mu} z_\lambda \prod_{i\geq 1} (1 - t^{\lambda_i})^{-1}.$$

Among the many interesting things encoded in this family of symmetric functions, one finds information about enumerative aspects of the lattice of subgroups of finite abelian p-groups, the irreducible characters of $GL_n(q)$, and the Kostka-Foulkes polynomials (a q-analogue of weight multiplicity for GL_n). See Chapters II–IV of Macdonald (1979).

2. The zonal polynomials Z_λ (see James (1961), Takemura (1984)). These are orthogonal relative to the form

$$\langle p_\lambda, p_\mu \rangle_2 := 2^{\ell(\lambda)} z_\lambda \delta_{\lambda\mu},$$

where $\ell(\lambda)$ is the length (number of nonzero terms) of λ. If we regard symmetric functions as functions of a matrix argument A by means of the identification $p_r = \mathrm{trace}(A^r)$, then the zonal polynomials are the spherical functions for the homogeneous space GL_n/O_n of $n \times n$ real, symmetric positive definite matrices.

3. The Jack symmetric functions $J_\lambda(\alpha)$ (see Stanley (1989)). These are orthogonal relative to the scalar product

$$\langle p_\lambda, p_\mu \rangle_\alpha := \alpha^{\ell(\lambda)} z_\lambda \delta_{\lambda\mu}.$$

One obtains the Schur functions if $\alpha = 1$, and the zonal polynomials if $\alpha = 2$.

4. Macdonald's two-parameter symmetric functions $M_\lambda(q,t)$ (see Macdonald (1988)). These are orthogonal relative to the scalar product

$$\langle p_\lambda, p_\mu \rangle_{q,t} := \delta_{\lambda\mu} z_\lambda \prod_{i\geq 1} \frac{1 - q^{\lambda_i}}{1 - t^{\lambda_i}},$$

and are of interest mostly because of what is conjectured about them [see especially Garsia and Haiman (1993)].

3. Some data structures

The following is a description of the special data structures employed by SF.

3.1. Partitions

Several of the procedures of SF are designed to accept partitions as input, or return partitions as output. For these purposes, a partition is defined to be a list of zero or more non-increasing positive integers. Thus `[3,3,2,1,1]` is a partition, but `[3,0,1]` is not. The empty partition is `[]`. It should be noted that Maple's built-in `combinat` package uses a different definition of 'partition'.

The SF package provides the procedures `Par`, `subPar`, and `dominate` for generating lists of partitions subject to various constraints.

3.2. Bases

Every basis of the ring of symmetric functions that SF understands has a Maple name reserved for it. There are four such bases that are defined when SF is first loaded. They and their names are:

- `e` , for the elementary symmetric functions;
- `h` , for the complete homogeneous symmetric functions;
- `p` , for the power-sum symmetric functions;
- `s` , for the Schur functions.

Note that the monomial symmetric functions m_λ are conspicuously absent from this list. However this is easily rectified using SF's mechanism for adding bases during a Maple session. This is discussed in more detail below.

The first three bases in the above list are distinguished by the fact that they are multiplicative. If `b` is the name of any such multiplicative basis (i.e., `b=e`, `h`, or `p`), then the generators of the basis `b` are denoted `b1,b2,b3,...` by SF, and the elements of the basis are monomials in these variables. Thus for example, the elementary symmetric functions $e_1, e_2, e_3, \ldots$ are expressed as `e1,e2,e3,...`, and if λ is the partition (3211), then e_λ can be expressed as `e3*e2*e1^2`. One implication of this is that the Maple names `e1,e2,e3,...` and `e` itself play a special role in SF, and thus should not be assigned values by the user. Similar remarks apply to `h` and `p`.

On the other hand, if `b` is the name of a non-multiplicative basis (e.g., `b=s`), then the element of this basis indexed by the partition $\lambda = (i_1, i_2, \ldots, i_k)$ is expressed by SF as the indexed variable `b[i_1,i_2,...,i_k]`. Thus for example, the Schur functions s_{3211}, s_5 and $s_\emptyset$ are expressed as `s[3,2,1,1]`, `s[5]` and `s[]`.

3.3. Symmetric Functions

A key part of the SF package are the four procedures `toe`, `toh`, `top`, and `tos` that take as input any symmetric function and convert it '`to`' an expression involving the bases e, h, p, and s, respectively.

For the purposes of SF, a *symmetric function* is defined to be any Maple expression that is a polynomial (i.e., of type `polynom`) with respect to the variables of all bases that have been defined. For example, `(1+q*h3)^2*s[2,2,1]-p2*s[4,2]^3` is a valid symmetric function, but `1/(1-p1)` is not. Of course, a symmetric function is also allowed to depend arbitrarily on other parameters; any variable that is not part of a defined

basis is assumed to belong to the field of scalars. It should be emphasized that SF *never* processes symmetric polynomials that are expressed explicitly in terms of the independent variables $x_1, \ldots, x_n$.† However, it is possible to recover the explicit dependence of a symmetric function on the x_i's using the procedure `evalsf`. Since the number of terms in a generic polynomial of this type is proportional to $n!$, any attempt to do arithmetic with these expressions would be doomed by exponential growth to work only with toy-sized problems.

In many of the procedures of SF, the user has the option of specifying as part of the input that a given symmetric function, say `f`, is expressed entirely in terms of some particular basis `b`. To be "expressed in terms of base `b`" means one of two things. First, if `b` is one of the non-multiplicative bases, it means that the only symmetric function variables that appear in the definition of `f` are of the form `b[i_1,...,i_k]`, where `[i_1,...,i_k]` ranges over all partitions, and that `f` is a linear expression with respect to these variables. Second, if `b` is one of the multiplicative bases, it simply means that the only symmetric function variables that appear in the definition of `f` are `b1,b2,b3,...`.

Thus for example, if `m` has been defined to be a (non-multiplicative) basis, then the *expression* `m[2,1]*m[3,1,1]` *is a valid symmetric function, but it is not considered to* be "expressed in base `m`." On the other hand, `m[3]+q*m[2,1]+q^3*m[1,1,1]` is a valid expression in base `m`.† However, the Schur function basis `s` is an exception to this. Even though it is non-multiplicative, the algorithms used to process expressions involving Schur functions (see the discussion in §4) are such that linearity need not be assumed.

3.4. Scalar Products

In SF, there are several procedures (see `dual_basis`, `add_basis`, `scalar`, and `skew`) that involve computations in the ring of symmetric functions relative to some user-specified scalar product. The most basic of these is `scalar`, which is designed to compute the scalar product of two symmetric functions. As we saw in §2, all of the useful scalar products on Λ share the property that the power sums p_λ are orthogonal. To specify such a product $(\ ,\ )$, the only additional information one needs is the value of (p_λ, p_λ) for all partitions λ.

For this reason, SF is designed so that the user specifies a scalar product by supplying a Maple procedure that accepts any partition λ as input, and returns as output the desired value of (p_λ, p_λ), the assumption being that $(p_\lambda, p_\mu) = 0$ for $\lambda \neq \mu$. Thus for example, to compute the scalar product of two symmetric functions, one would call `scalar` with three arguments—the first two being the symmetric functions, and the third being (the

† 2
† 3

name of) the procedure that computes (p_λ, p_λ). To be more specific, let us recall that the standard scalar product is defined by the property that $\langle p_\lambda, p_\lambda \rangle = z_\lambda$ (see (1)). One of the basic procedures of SF is **zee**, which returns the value of z_λ for each partition λ. Hence to compute the standard scalar product of symmetric functions **f** and **g**, one could use the command `scalar(f,g,zee)`. One could also compute this scalar product more simply via the command `scalar(f,g)`, since if the scalar product specification is omitted, the default is to use the standard one.

We should also point out that **zee** is designed to accept one or two additional optional arguments which facilitate the specification of scalar products other than the standard one. For example, to specify the scalar product $\langle , \rangle_t$ used in the definition of Hall-Littlewood symmetric functions (see §2), one could use

```
zee_for_hall_littlewood:= lambda -> zee(lambda,0,t).
```

3.5. Adding New Bases

There are two procedures in SF, **dual_basis** and **add_basis**, that are designed to allow the user to define new symmetric function bases to add to the repertoire of SF. The first of these, **dual_basis**, takes any previously defined basis b_λ, together with a user-specified scalar product (,), and adds to SF the basis that is dual to b_λ; i.e., the unique basis B_λ satisfying $(B_\lambda, b_\mu) = \delta_{\lambda\mu}$. The precise calling sequence is

```
dual_basis(B,b,scp);
```

where **B** is a user-chosen name for the dual basis to be created, **b** is the name of the previously defined basis, and **scp** is the scalar product; i.e., the procedure for computing (p_λ, p_λ). If the scalar product is omitted, the default is to use the standard one.

The net effect of such a procedure call is the following. First, a new procedure, named **toB**, is defined. Its purpose is similar to the built-in procedures **toe**, **toh**, **top** and **tos**: it converts any given symmetric function into an expression in the base **B**. Second, all previously defined '**to**'-procedures are provided with information sufficient to allow them to correctly process symmetric function expressions involving the new basis **B**.

For example, an obvious use for this feature is to define the monomial symmetric functions m_λ. Recall from (1) that the m-basis is dual to the h-basis, relative to the standard scalar product. Thus to define the m-basis in SF, one simply uses `dual_basis(m,h)`. At this point, a new procedure **tom** will be created, and the user is free to process symmetric function expressions involving the variables of the m-basis.

The second mechanism for defining new bases is **add_basis**. This procedure is designed for symmetric function bases, such as those discussed in §2, that can be characterized by orthogonality-plus-triangularity properties. More precisely, let us suppose that (,) is a scalar product on Λ for which $(p_\lambda, p_\mu) = 0$ for $\lambda \neq \mu$. This given, there is a unique basis

P_λ of Λ that is orthogonal relative to (,), and satisfies

$$P_\lambda = c_\lambda m_\lambda + \text{ a linear combination of } m_\mu \text{ for } \mu < \lambda,$$

where c_λ is any (specified) nonzero scalar, and '$\mu < \lambda$' refers to any fixed total ordering of the set of partitions.

If `scp` is the Maple procedure for computing (p_λ, p_λ), and if `lterm` is a Maple procedure that computes the desired "leading term" c_λ for each partition λ, then the procedure call

```
add_basis(P,scp,lterm);
```

defines `P` to be the name of the symmetric function basis characterized by the above properties, using a lexicographic total ordering on the set of partitions. If the third argument, `lterm`, is omitted, the default is to use $c_\lambda = 1$. The effect of `add_basis` is similar to `dual_basis`: a new procedure named `toP` is defined, and all previously defined '`to`'-procedures are informed of the existence of the new basis `P`.

3.6. Characters

There is a close relationship between characters of the symmetric group S_n and the space of homogeneous symmetric functions of degree n; several of the procedures of `SF` (e.g., `itensor`, `plethysm`, `scalar`, `skew` and `tos`) have special significance in this context. (For details of the mathematical theory behind this, see James and Kerber (1981), Littlewood (1950), or Macdonald (1979).) Furthermore, there are two procedures, `sf2char` and `char2sf`, that are designed to convert between symmetric functions and the symmetric group characters to which they correspond. For these purposes, a symmetric group character is defined to be a linear combination of Maple expressions of the form `cl[i_i,...,i_k]`, where `[i_i,...,i_k]` ranges over all partitions. Here '`cl`' is a special name reserved by `SF` to designate class functions on the symmetric group; the expression `cl[i_i,...,i_k]` represents a function on the symmetric group that takes on the value 1 at any permutation of cycle-type $\lambda = (i_i, \ldots, i_k)$, and 0 otherwise. Thus for example, the Maple expression

```
3*cl[1,1,1,1]+cl[2,1,1]-cl[2,2]-cl[4]
```

represents the (potentially virtual) character of S_4 whose values are $3, 1, -1$, and -1 on permutations of cycle-type $(1111), (211), (22)$, and (4), respectively, and 0 otherwise.

4. Some Remarks About the Algorithms

At the heart of the package are the algorithms for converting an arbitrary symmetric function expression into an expression in terms of some particular basis. Although there exist numerous elaborate combinatorial rules for carrying out basis conversions, all of the algorithms employed by this package are algebraic and symbolic.

Among these algorithms, the most crucial are the ones involving conversions between any two of the bases e, h, and p, since all of the other basis conversion algorithms, as well as some of the procedures for performing operations on symmetric functions, call these as subroutines. Conversions between polynomial expressions involving the variables e_r, h_r, and p_r are theoretically easy, given the existence of the convolution identities

$$\begin{aligned}\sum_{0\le r\le n}(-1)^r e_r h_{n-r} &= 0,\\ \sum_{0<r\le n} p_r h_{n-r} &= n h_n,\\ \sum_{0<r\le n}(-1)^{r-1} p_r e_{n-r} &= n e_n,\end{aligned}$$

for all $n \ge 1$. However in practice, one finds that if the above relations are recursively applied to a generic expression of degree roughly 20, the space consumed during conversion to a normal form is prohibitive. We instead use the following non-recursive approach. The generating functions $E(t) = \sum_{n\ge 0} e_n t^n$, $H(t) = \sum_{n\ge 0} h_n t^n$, and $P(t) = \sum_{n\ge 1} p_n t^n$ are easily shown to satisfy the relations

$$E(-t)H(t) = 1 \quad \text{and} \quad P(t) = t\frac{d}{dt}\log H(t).$$

Thus one can express any one of the three generating functions in terms of any one of the others. Suppose for example that an expression f in the basis e is to be converted to the basis h. Our algorithm proceeds by first finding the largest integer n such that e_n occurs in f. It then determines the Taylor series expansion for

$$E(t) = H(-t)^{-1} = 1/(1 - h_1 t + \cdots + (-1)^n h_n t^n) + O(t^{n+1}),$$

through terms of order t^n, using a built-in Maple procedure. It then replaces each occurrence of e_r in f with the coefficient of t^r in the Taylor series, and converts the result to a normal form.

For conversions involving the basis of Schur functions, the algorithms used are as follows. First, given an expression f involving Schur functions, each occurrence of s_λ is replaced with one of the determinants

$$s_\lambda = \det[h_{\lambda_i - i+j}] = \det[e_{\lambda'_i - i+j}],$$

where λ' denotes the conjugate partition. The size of the first determinant is the number of rows in the Young diagram of λ, whereas for the second it is the number of columns. Thus for symmetric functions of degree n this conversion can be accomplished with determinants of order at most $n/2$. The result is then a mixed expression involving the h- and e-bases that can be converted to an expression in any desired basis. Conversely,

for conversion of a homogeneous symmetric function f into a linear combination of Schur functions, the algorithm proceeds by first converting f into an expression in the h-basis, and then the degree of f is computed, first with respect to the (usual) grading in which $\deg(h_r) = r$, and second with respect to the grading in which $\deg(h_r) = 1$. Assuming the former degree is n and the latter is d, then one knows *a priori* that there exist scalars c_λ such that

$$f = \sum_\lambda c_\lambda s_\lambda,$$

where λ ranges over the partitions of n with at most d parts. Regarding the variables c_λ as indeterminates, the algorithm then converts the identity $f = \sum c_\lambda s_\lambda$ into an equation in the h-basis as above. The coefficients of the result yield linear equations that determine the c_λ's. To improve the performance of the algorithm, the user is also allowed to specify in advance any set of partitions that is known to contain the "support" of f (i.e., those λ such that $c_\lambda \neq 0$).

For conversions involving a basis b created by `dual_basis`, the algorithms are as follows. Let b^* denote the basis that b is defined to be dual to. Given a homogeneous symmetric function f of degree n in the basis b, the algorithm chooses indeterminates c_λ for each partition of n, and then converts the symbolic linear combination

$$g := \sum_\lambda c_\lambda p_\lambda / (p_\lambda, p_\lambda),$$

into an expression in the b^*-basis. The p-expansion of f is obtained by evaluating

$$\sum_\lambda g|_{b^*_\lambda} \cdot f|_{b_\lambda},$$

and then applying the substitution $c_\lambda \mapsto p_\lambda$. (Here we are using the notation $f|_{b_\lambda}$ to denote the coefficient of b_λ in f.) The result can now be converted to an expression in any other desired basis.

Conversely, for conversions into the basis b, one could determine the coefficient of b_λ in an arbitrary symmetric function f by computing (b^*_λ, f). However, we have found in practice that it is more efficient to simultaneously compute every needed coefficient with just one call to the subroutine for evaluating $(\ ,\ )$, as follows. Let $g := \sum c_\lambda b^*_\lambda$, where the c_λ's are indeterminates and the sum ranges over the partitions of $n = \deg(f)$ (or some smaller set known to contain the b-support of f, optionally supplied by the user). We obtain the b-expansion of f by evaluating (g, f) and applying the substitution $c_\lambda \mapsto b_\lambda$.

For conversions involving a basis P created by `add_basis`, the algorithms are as follows. First, given any expression f involving the symmetric functions P_λ, the strategy is to substitute the e-basis expansion for each P_λ appearing in f. The result is then converted into an expression in any desired basis. The algorithm for computing the e-expansion of

P_λ proceeds by writing P_λ as a symbolic linear combination

$$P_\lambda = c_\lambda e_{\lambda'} + \sum_{\mu<\lambda} a_\mu e_{\mu'},$$

where c_λ is the user-specified leading term for P_λ, the sum ranges over partitions μ that precede λ in lexicographic order, and the a_μ's are indeterminates. That such an expression for P_λ exists is a consequence of the fact that the transition matrix between the basis $e_{\lambda'}$ and m_λ is triangular with a unit diagonal. The coefficients a_μ are determined by solving the triangular system of linear equations $(P_\lambda, e_{\mu'}) = 0$ for $\mu < \lambda$. The solution is stored in a Maple remember table so that subsequent calculations involving P_λ require only a table look-up. We use the e-basis in these calculations, instead of the m- or p-bases, for several reasons. First, computations involving the e-basis are typically faster than with the m-basis. Second, we have found that for the most common bases definable via `add_basis`, the coefficients relative to the e-basis require significantly less storage than those relative to the m- or p-bases. Third, the transition matrix between the P-basis and the e-basis is triangular, whereas between the P-basis and the p-basis, most entries of the matrix are nonzero. This saves another factor of two in storage space.

Conversely, for conversion of a (homogeneous) symmetric function f into an expression in the basis P_λ, the algorithm proceeds by first converting f into an expression in the e-basis. It then computes the degree of f with respect to the grading $\deg(e_r) = r$ and the grading $\deg(e_r) = 1$. Assuming that these two degrees are n and d, respectively, then one may write f as a linear combination

$$f = \sum_\lambda a_\lambda P_\lambda,$$

where λ ranges over the partitions of n with parts $\leq d$ (or some smaller set optionally supplied by the user) and the a_λ's are indeterminates. The next step is to convert the right side of the above identity into an expression in the e-basis, using the previous algorithm. After collecting coefficients (relative to the e-basis), one obtains a system of linear equations that determine the a_λ's.

To temper the user's enthusiasm, let us point out that several of the symmetric function bases discussed in §2 depend on various free parameters such as q, t, and α. In such cases, the field of scalars for the algorithms is a rational function field. The complexity of arithmetic over such fields and the size of the intermediate expressions involved can severely test the capacity of the hardware. To give some measure of what to expect, the following are some benchmarks on a Sparc 10/41 running *Maple V* (Release 1). It takes about 45sec and 1.4 MB to build the remember table for the Hall-Littlewood functions for

all partitions of 6; for partitions of 8, it takes about 14.5min and 2.6MB. For Macdonald's symmetric functions and partitions of 6, it takes about 10min and 2.7MB.†

5. Short synopses of the procedures

The following is a brief indication of the purpose of each procedure in SF. The full details, including syntax, definitions and examples, are provided in the on-line help.

`Par`	list partitions
`subPar`	list all subpartitions of a partition
`dominate`	list all partitions dominated by another partition
`conjugate`	conjugate a partition
`hooks`	hook lengths of a partition
`toe`	convert a symmetric function to the e-basis
`toh`	convert a symmetric function to the h-basis
`top`	convert a symmetric function to the p-basis
`tos`	convert a symmetric function to the Schur function basis
`add_basis`	add a new basis to the set of known bases
`dual_basis`	add a dual basis to the set of known bases
`itensor`	inner tensor product of symmetric functions
`omega`	apply the omega automorphism to a symmetric function
`theta`	apply the theta automorphism to a symmetric function
`plethysm`	plethysm of symmetric functions
`skew`	skew operation on symmetric functions
`scalar`	scalar product of symmetric functions
`zee`	squared norm of power sums
`char2sf`	convert a (virtual) character into a symmetric function
`sf2char`	convert a symmetric function into a (virtual) character
`evalsf`	evaluate (or specialize) a symmetric function
`jt_matrix`	Jacobi-Trudi matrix of a (possibly skew) partition
`varset`	variable set (or "support") of a symmetric function

The following is a short summary of the contents of the `examples` subdirectory. These files must be explicitly loaded by the user.

`bases`	annotated list of SF-definitions for commonly used symmetric functions
`graphs`	count nonisomorphic graphs via the Polya-Redfield method
`macdonald`	compute Macdonald's symmetric functions
`m_conjecture`	test Macdonald's conjecture regarding the q,t-Kostka matrix
`qt_kostka`	compute the entries of the q,t-Kostka matrix
`seminormal`	generate matrices for Young's seminormal representations of S_n
`tableaux`	generate all standard Young tableaux of a given shape
`LR_rule`	implementation of the Littlewood-Richardson rule

† However, one can exploit some special properties of Macdonald's symmetric functions in a way that greatly accelerates the computation. Some Maple code for this is provided in the `examples` directory of SF. Using this specially designed code, the same calculation takes about 85sec and 1.9MB. For partitions of 10, it takes about 18hrs and 21MB.

References

Fulton, W. and Harris, J. (1991). *Representation Theory. A First Course*, Springer-Verlag, New York.

Garsia, A. M. and Haiman, M. (1993). A graded representation model for Macdonald's polynomials, *Proc. Nat. Acad. Sci. U.S.A.* **90**, 3607–3610.

James, A. T. (1961). Zonal polynomials of the real positive definite symmetric matrices, *Ann. of Math.* **74**, 475–501.

James, G. D. and Kerber, A. (1981). *The Representation Theory of the Symmetric Group*, Addison-Wesley, Reading, MA.

Littlewood, D. E. (1950). *The Theory of Group Characters*, 2nd ed., Oxford Univ. Press, Oxford.

Macdonald, I. G. (1979). *Symmetric Functions and Hall Polynomials*, Oxford Univ. Press, Oxford.

Macdonald, I. G. (1988). A new class of symmetric functions, *Publ. I.R.M.A. Strasbourg, Actes 20e Séminaire Lotharingien*, 131–171.

Sagan, B. E. (1991). *The Symmetric Group. Representations, Combinatorial Algorithms, and Symmetric Functions*, Wadsworth & Brooks/Cole, Monterey.

Stanley, R. P. (1971). Theory and application of plane partitions. Parts 1 and 2. *Stud. Appl. Math.* **50**, 167–188, 259–279.

Stanley, R. P. (1989). Some combinatorial properties of Jack symmetric functions, *Adv. in Math.* **77**, 76–115.

Takemura, A. (1984). *Zonal Polynomials*, Institute of Mathematical Statistics, Hayward, CA.