A Maple Package for Symmetric Functions

Version 2.4[*]

9 November 2005

John R. Stembridge[1]
Department of Mathematics
University of Michigan
Ann Arbor, Michigan 48109–1043 USA

email: `jrs@umich.edu`
`www.math.lsa.umich.edu/~jrs`

Contents

1. Introduction

This document provides an introduction to Version 2.4 of `SF`, a package of 25 Maple programs that create an environment for computations involving symmetric functions and related structures. Included are programs for manipulating partitions, for converting symmetric functions between various bases, for applying standard operations such as plethysm, tensor (Kronecker) and scalar products, as well as procedures for adding new bases to the package. These features make `SF` especially useful for applications involving

- characters of S_n and GL_n,
- classical invariant theory,
- Hall-Littlewood functions and Kostka-Foulkes polynomials,
- zonal polynomials for GL_n/O_n, and
- Macdonald's two-parameter symmetric functions.

What's New. Version 2.4 of `SF` is a major rewrite of Version 2.3, but remains upward-compatible with it. The main benefit of this rewrite is that the high-level functions are

[*] Version 2.4 is the Hermann Weyl Birthday Edition.

[1] This work has been supported by NSF grants DMS–0070685 and DMS–0245385.

dramatically faster and more space-efficient than the old ones. For example, converting a generic symmetric function of degree 30 with rational coefficients from power-sums to elementary symmetric functions takes just a few GHz-seconds[2] in SF 2.4, about 7 to 17 times faster than SF 2.3 (depending on the Maple version). Computing the Schur function decomposition of the plethysm $s_5[s_6]$ is about 4 to 6 times faster (roughly 10 GHz-seconds), and decomposing p_1^8 into Hall-Littlewood functions is about 35 to 45 times faster.

A few of the procedures now support new options, and there is a new procedure `nextPar` for constructing space-efficient loops through the set of partitions.

For a more detailed account of the new features and improvements, see §7.

I still get reports from users that have obsolete versions of SF, including versions from the Maple Share Library. These users should upgrade and take advantage of the vast improvements in the package that have been added since Version 2.0.

2. System Requirements

First, you must have some version of Maple ⟨`http://www.maplesoft.com`⟩.

This version of SF has been developed and tested on various Linux machines running Releases 3, 4, and 5 of Maple V, Maple 6, Maple 7, Maple 9, and Maple 9.5. While I no longer have access to machines running the first two releases of Maple V, and have not yet tried Maple 10, the package will probably work with these versions as well.

Version 2.4 is packaged in two formats: one is a Unix `tar` file (the "Unix Edition"), and the second is a collection of plain text files that comprise the Vanilla Edition. Both editions are available for download from

⟨`http://www.math.lsa.umich.edu/~jrs/maple.html`⟩.

The minor functional differences between these two editions are discussed in §3. Users of Maple on Unix platforms have the choice of installing either edition of SF. For everyone else, the Vanilla Edition is the only choice.

3. Getting Started

Special Notes for the Unix Edition. The first step is to unpack the `tar` file and follow the installation instructions in the provided `READ_ME` file. If a system administrator has done the installation for you, and you aren't already a user of one of my packages, then you will need to find out the name of the directory where the package has been installed and edit an initialization file.

For the purposes of this discussion, let's assume that the SF package has been installed in the directory `/usr/local/maple/packages`. You can check this by verifying that `/usr/local/maple/packages` has a subdirectory named SF. Next, you will need to create a file named `.mapleinit` in your home directory. Or add to it, if you already have one. In this file you should insert the following two lines:

```
HomeLib:=`/usr/local/maple/packages`:
libname:=libname, HomeLib:
```

[2] A GHz-second represents 1 second of CPU time on a 1 GHz processor.

Be careful to correctly type both of the left quote-marks (`` ` ``). Each time Maple is invoked, these statements will define where the package is located. (For Maple V Release 1, the syntax is slightly different—see the notes below.)

You are now ready to run `SF`. Simply enter the command `with(SF)` during a Maple session. To access documentation about an individual function, use `?SF[`*function*`]` or `?SF,`*function*. The source code for `SF` is located in

```
/usr/local/maple/packages/SF/src,
```

and the raw text for the on-line help is located in

```
/usr/local/maple/packages/help/SF.
```

Additional files that provide applications and extensions of `SF` can be found in

```
/usr/local/maple/packages/SF/examples.
```

Special Notes Regarding Maple V Release 1. The syntax for specifying the locations of external libraries is slightly different in Maple V Release 1. The `.mapleinit` file should instead have the lines

```
HomeLib:=`/usr/local/maple/packages`:
_liblist:=[HomeLib]:
```

Be careful to correctly type the left quote-marks (`` ` ``) and underscore character (`_`).

Also, when the Unix Edition of `SF` is installed with Release 1 of Maple V, it is no longer the case that the help texts are integrated into the package. Instead, users should consult the HTML version of the help texts provided with the Vanilla Edition. (The mechanism for loading help for external functions is so primitive in Release 1 that it no longer makes sense to support it in `SF`.)

Special Notes for the Vanilla Edition. The Vanilla Edition consists primarily of a single text file that may be loaded into a Maple session via the `read` command. It loads all of the package functions into the workspace, whereas the Unix Edition keeps clutter to a minimum by loading functions at the moment they are first used.

When the Vanilla Edition is first loaded, the package functions may be used only via the long form

`SF[`*function*`](`*arguments*`).`

Although `SF` is not recognized by the Maple `with` command, the Vanilla Edition provides a special command `withSF()` that enables the use of the short forms

function`(`*arguments*`).`

Similarly, `withSF(`*arguments*`)` enables the short form for some subset of the package, modeling the expected behavior of `with(SF,`*arguments*`).`

Documentation for the individual functions is not loaded when the package is loaded. Instead there is a separate HTML file containing all of the help texts cross-referenced with hyperlinks. Similarly, the suite of examples is provided as a directory of plain text files available for download.

The remainder of this article presents an overview of the workings of the package, explaining the fundamental data structures and basic definitions. A one-line description of the purpose of each procedure is provided in §6. For more detailed information about the individual commands, consult the on-line help.

4. Some Definitions

We begin with a brief review of some basic ingredients of the theory of symmetric functions. It is highly recommended, but not strictly necessary, that the reader be acquainted with the definitive treatment by Macdonald [**7**]. Other reasonable places to look are [**1**] (Appendix A), [**6**], [**9**], and [**11**] (Chapter 7), 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 polynomial 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 there are sufficiently many of them. Taking this idea to its logical extreme, one is led to consider symmetric "polynomials" in infinitely many variables, say $x_1, x_2, \ldots$.[3] These symmetric pseudo-polynomials, also known as symmetric functions, form a graded ring Λ. (For a mathematically rigorous definition, see [**7**].)

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

$$e_r = \sum_{i_1 < i_2 < \cdots < i_r} x_{i_1} \cdots x_{i_r};$$

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

$$h_r = \sum_{i_1 \leq i_2 \leq \cdots \leq i_r} x_{i_1} \cdots x_{i_r};$$

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

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

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 for Λ. Since e_r is homogeneous of degree r, it follows that the dimension of the nth graded component of Λ is $p(n)$, the number of partitions of n into a sum of positive integers (disregarding order). If $n = \lambda_1 + \cdots + \lambda_l$ is such a partition, let us 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.

[3] Strictly speaking, these are no longer polynomials, but formal power series.

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 bases of Λ of fundamental importance, the monomial symmetric functions m_λ and the Schur functions s_λ. For the monomial symmetric functions, given a partition $\lambda = (\lambda_1, \ldots, \lambda_l)$, one has

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

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 [**1**], [**5**], or [**7**].)

There is one obvious distinction between the first three bases we defined (e_λ, h_λ, and p_λ) and the last two (m_λ and s_λ): the latter are not *multiplicative.* That is, 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 natural scalar product on Λ that is (over-)determined by the properties

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

where $z_\lambda := a_1!\, 1^{a_1} a_2!\, 2^{a_2} \cdots$, using a_i to denote the number of copies of i in λ. In connection with this definition/assertion it is important to be aware that the transition matrix between Schur functions and monomial symmetric functions is triangular. In fact,

$$s_\lambda = \sum_{\lambda \ge \mu} K_{\lambda,\mu} m_\mu \tag{2}$$

for certain scalars $K_{\lambda,\mu}$ (the Kostka numbers), where the notation '$\lambda \ge \mu$' indicates that λ and μ must be related in the "dominance" partial order; i.e.,

$$\lambda_1 + \cdots + \lambda_i \ge \mu_1 + \cdots + \mu_i \text{ for all } i \ge 1.$$

A consequence of (1) and (2) is that the Schur functions are the unique orthonormal basis of Λ (relative to $\langle\ ,\ \rangle$) that one would obtain by linearly ordering the m_λ's in any way consistent with the dominance order and applying the Gram-Schmidt algorithm. Alternatively, using the fact that $K_{\lambda,\lambda} = 1$, one could 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. Indeed, there are a number of bases in the literature on symmetric functions that may be characterized (up to scalar multiples) by (i) orthogonality with respect to some scalar product, and (ii) having a triangular transition matrix with respect to the monomial symmetric functions in dominance order.

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

1. The Hall-Littlewood functions $HL_\lambda(t)$ (see Chapters II–IV of [**7**]). 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: (i) counting chains in the lattice of subgroups of finite abelian p-groups, (ii) the irreducible characters of $GL_n(q)$, and (iii) the Kostka-Foulkes polynomials (a q-analogue of the Kostka numbers).

2. The zonal polynomials Z_λ (see [**4**], [**12**], and Chapter VII of [**7**]). These are orthogonal relative to the scalar product

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

where $\ell(\lambda)$ is the length (number of nonzero terms) of λ. If we specialize to n variables $x_1, \ldots, x_n$, we obtain (after agreeing about questions of normalization) the spherical functions for the homogeneous space GL_n/O_n of $n \times n$ real, symmetric positive definite matrices. (One identifies $x_1, \ldots, x_n$ as the eigenvalues of a positive definite matrix.)

3. The Jack symmetric functions $J_\lambda(\alpha)$ (see [**10**] and §VI.10 of [**7**]). 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 $P_\lambda(q,t)$ (see [**8**] and Chapter VI of [**7**]). 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 have stimulated a vast amount of research (e.g., see [**2**] and [**3**]).

5. Some Data Structures

There are several data structures that play special roles in `SF`.

Partitions. Many of the procedures in `SF` are designed to accept partitions as input, or return partitions as output. For these purposes, a partition is defined to be a non-increasing list of zero or more positive integers. Thus `[3,3,2,1,1]` is a partition, but `[3,1,0]` and `[1,1,2]` are not. The empty partition is `[]`. Note that Maple's built-in `combinat` package arranges the parts of partitions in non-decreasing order.

The procedures `Par`, `subPar`, and `dominate` are useful for generating lists of partitions satisfying various constraints.

Partition Orderings. There are two total orderings of the set of partitions that are important in `SF`. The lexicographic ordering[4] is defined by the rule that $\lambda >_L \mu$ if

$$\lambda_1 = \mu_1,\ \ldots,\ \lambda_{i-1} = \mu_{i-1},\ \lambda_i > \mu_i \ \text{ for some } i \geq 1,$$

and the conjugate-lexicographic ordering is defined by

$$\lambda <_C \mu \quad \Leftrightarrow \quad \lambda' >_L \mu',$$

where λ' denotes the partition conjugate to λ; i.e., λ'_i is the number of parts $\geq i$ in λ.

These two orderings are quite similar, but note that for partitions of 6, we have

$$6 >_L 51 >_L 42 >_L 411 >_L 33 >_L 321 >_L 3111 >_L 222 >_L 2211 >_L 21111 >_L 111111,$$
$$6 >_C 51 >_C 42 >_C 33 >_C 411 >_C 321 >_C 222 >_C 3111 >_C 2211 >_C 21111 >_C 111111.$$

Both orderings are refinements of the dominance order.

The procedure `Par` lists partitions of n in lexicographic order, starting with `[n]` and ending with `[1,...,1]` (assuming $n > 0$). Given a partition `lambda` of n, the procedure call `nextPar(lambda)` generates the next partition that appears after `lambda` in the list `Par(n)` (or `NULL` if there is no such partition), and thus may be used to construct space-efficient partition loops. For example,

```
lambda:=[30];
while lambda <> NULL do
  f(lambda):
  lambda:=nextPar(lambda):
end do:
```

would apply the function `f` to each partition of 30.

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

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

Note that the monomial symmetric functions m_λ are conspicuously absent from this list. However this is no great loss, since it is easy to add more bases (including the m_λ's) to the repertoire of `SF`. See the discussion below regarding "Adding New Bases."

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

[4] We can think of at least four total orderings that could be called "lexicographic."

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 may 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[]`. Note that if `lambda` is a partition, say `lambda:=[3,3,1]`, then the Maple expression `b[lambda]` evaluates to `b[[3,3,1]]`, which is not a proper expression for an element of the basis `b`. Instead one should write `b[op(lambda)]`.

The `SF` package uses a global variable, `` `SF/Bases` ``, to track the set of symmetric function bases that have been defined in the current Maple session. In particular, when `SF` is first loaded, the assignment

```
`SF/Bases`:={e, h, p, s[]}
```

is used to initialize this variable. The reason that '`s[]`' appears here, rather than '`s`', is to indicate that the Schur function basis is not multiplicative.

Symmetric Functions. At the heart of the `SF` package are 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 each multiplicative basis `b`, these variables are `b1,b2,...`, and for each non-multiplicative basis `b` these variables are `b[i_1,...,i_k]`, where `[i_1,...,i_k]` ranges over all partitions. Of course, a symmetric function is also allowed to depend arbitrarily on other parameters. It should be emphasized that `SF` *never* processes symmetric polynomials that are expressed explicitly in terms of dependent variables, such as $x_1, \ldots, x_n$.[5] 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.

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. Note also that unless '`Z`' has already been defined as the name of a symmetric function basis, `SF` will assume that variable names such as `Z[2,2,1]` belong to the field of scalars.

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: (1) 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.

[5] However, using the procedure `evalsf`, it is possible to recover the explicit dependence of a symmetric function on the x_i's.

(2) 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]+(1+q^2)*m[1,1,1]` is a valid expression in base `m`.

Since the Schur function basis `s` is non-multiplicative, similar remarks ought to apply to it as well. However, due to the special nature of the algorithms we use to convert between Schur functions and other symmetric functions, we do not need to impose linearity on expressions designated to be expressed in base `s`. Hence, in contrast to the previous example, `SF` would tolerate being informed that `s[2,1]*s[3,1,1]` is in base `s`.

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 §4, all of the useful scalar products on Λ share the property that the power-sums p_λ are orthogonal. To specify such a product, say $(\ , \ )$, the only additional information one would need is the value of (p_λ, p_λ) for all partitions λ.

For these reasons, `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 name of the procedure that computes (p_λ, p_λ). To be more specific, 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 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 may also compute this via the command `scalar(f,g)`, since the default is to use the standard scalar product if none is specified.

It should be noted that `zee` is designed to accept one or two additional optional arguments that facilitate the specification of scalar products other than the standard one. For example, the scalar product $\langle \ , \ \rangle_t$ used in the definition of Hall-Littlewood symmetric functions (see §4) could be specified by the procedure

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

Adding New Bases. There are two procedures in `SF`, `dual_basis` and `add_basis`, that are designed to allow the user to introduce new symmetric function bases into `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, the name `B[]` is added to `` `SF/Bases` ``, the global variable that keeps track of all bases that have been defined. Second, a new procedure named `toB` is (silently) created. 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 `B`-basis. The third effect is that the information defining this basis is saved in tables so that all previously defined '`to`'-procedures are able to process symmetric function expressions involving the new basis `B`.

For example, probably the most common use of this feature is to define the monomial symmetric functions m_λ. Recall from (1) that the m_λ's are dual to the h_λ's, relative to the standard scalar product. Thus to define the m_λ's in `SF`, one simply uses `dual_basis(m,h)`. At this point, the new procedure `tom` will be created, and the user is free to process symmetric function expressions involving the monomials `m[i_1,...,i_k]`.

The second mechanism for introducing new bases is `add_basis`. This procedure is designed for symmetric function bases, such as those discussed in §4, that may 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 $\lambda \neq \mu$. Given scalars $c_\lambda \neq 0$, there is a unique basis P_λ of Λ that (i) is orthogonal relative to $(\ ,\ )$, and (ii) satisfies

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

where '$<_C$' denotes the conjugate-lexicographic ordering we defined earlier.

We remark that for $n \geq 1$, the P_λ's indexed by partitions λ with $\leq n$ parts specialize to a basis for the ring of symmetric polynomials in n variables. This is a special feature of the $<_C$-ordering and would not be valid in general if we had used the lexicographic ordering $<_L$ in our definition.[6]

If `scp` and `lterm` are Maple procedures for computing the scalar product (p_λ, p_λ) and 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 properties (i) and (ii) above. 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`: the name `P[]` is added to `` `SF/Bases` ``, a new procedure named `toP` is created, and the defining information for the `P`-basis is saved in tables so that all previously defined '`to`'-procedures are able to process symmetric function expressions involving the new basis `P`.

In principle, the user should not need to be concerned about how `SF` internally processes symmetric function expressions involving a basis created via `add_basis`; however, in order to give a rough idea of the resources that are needed during such computations, a few remarks about the algorithms are in order. Suppose that `P` is the name of a basis created by `add_basis`. On the first occasion that `SF` is asked to process a symmetric function involving, say `P[3,3,2,1]`, the first step is to use the Gram-Schmidt algorithm to compute the expansion of `P[3,3,2,1]` in terms of elementary symmetric functions, and

[6] However, it should be noted that the orthogonal bases that occur in applications, including all of the bases discussed in §4, are not sensitive to how the partitions are ordered, provided that the ordering is a refinement of the dominance order.

cache the result in a Maple remember table. Given the recursive nature of the Gram-Schmidt algorithm, this means that the expansions of all "earlier" members of the basis (e.g., `P[3,2,2,2]`, `P[3,2,2,1,1]`, etc.) will need to be computed first, or retrieved (if available) from a remember table. Thus, the first computation involving `P[3,3,2,1]` may be slow, but subsequent computations will require only a quick table lookup.

To temper the user's enthusiasm, note that several of the symmetric function bases discussed in §4 depend on various free parameters such as q, t, and α. In such cases, the Gram-Schmidt algorithm is forced to work over rational function fields. The complexity of arithmetic over such fields and the sheer size of the expressions involved mean that the adventurous user will severely test the capacity of Maple. For example, here are some benchmarks I obtained while running the Linux/x86 version of Maple V Release 5. It takes about 5.6 GHz-sec and 3MB to build the remember table for the Hall-Littlewood functions of degree 9; for degree 12, it takes about 2 GHz-min and 7MB. For the Macdonald symmetric functions of degree 8, it takes about 2 GHz-min and 13MB.[7]

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 about this, see [**5**], [**6**], or [**7**].) 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_1,...,i_k]`, where `[i_1,...,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_1,...,i_k]` represents a function on the symmetric group that takes on the value 1 at any permutation of cycle-type $\lambda = (i_1, \ldots, i_k)$, 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 (possibly virtual) character of S_4 whose values are $3, 1, -1$, and -1 on permutations whose respective cycle-types are $(1111), (211), (22)$, and (4).

6. Short Synopses of the Procedures

Here is a brief indication of the purpose of each procedure in `SF`. For the full details, including syntax, definitions and examples, consult the on-line help via the commands `?SF,`*function* or `?SF[`*function*`]`, or use the HTML document in the Vanilla Edition.

`Par`	list partitions
`add_basis`	add an orthogonal basis to the set of known bases
`char2sf`	convert (virtual) characters to symmetric functions
`conjugate`	conjugate a partition
`dominate`	list/test partitions dominated by another partition
`dual_basis`	add a dual basis to the set of known bases

[7] However, by exploiting special features of Macdonald's symmetric functions it is possible to greatly accelerate 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 18 GHz-sec and 6.6MB.

`evalsf`	plethystic evaluation of symmetric functions
`hooks`	hook lengths of a partition
`itensor`	inner tensor product of symmetric functions
`jt_matrix`	Jacobi-Trudi matrix of a (possibly skew) partition
`nextPar`	generate the next partition in lexicographic order
`omega`	apply the automorphism ω to a symmetric function
`plethysm`	plethysm of symmetric functions
`scalar`	scalar product of symmetric functions
`sf2char`	convert symmetric functions to (virtual) characters
`skew`	skew operation on symmetric functions
`stdeg`	degree with respect to the standard grading
`subPar`	list all subpartitions of a partition
`theta`	apply the automorphism θ to a symmetric function
`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
`varset`	variable set of a symmetric function
`zee`	squared norm of power sums

Here is a short summary of the contents of the `examples` subdirectory. These files must be explicitly loaded by the user and contain both documentation and source code.

`bases`	annotated list of `SF`-definitions for commonly used symmetric functions
`graph`	count unlabeled graphs via the Polya-Redfield method
`jacks`	accelerated computation of Jack symmetric functions
`kfpoly`	generate Kostka-Foulkes polynomials via rigged configurations
`LR_rule`	implementation of the Littlewood-Richardson rule
`macdonald`	accelerated computation of Macdonald's symmetric functions
`qt_kostka`	compute the entries of the q, t-Kostka matrix
`seminormal`	generate matrices for Young's seminormal representations of S_n
`skew_shapes`	generate all skew shapes of a given size
`tableaux`	generate all standard Young tableaux of a given shape

7. Changelog

Version 2.4. The most significant improvements and additions are:

- Conversions among the `e`, `h`, and `p` bases are more space-efficient and *much* faster.
- Calls to `` `solve/linear` `` (now deprecated in Maple) have been eliminated.
- The procedure `tos` for converting to Schur functions has been radically redesigned and is significantly faster. In particular, the Schur function support of a symmetric function is now inferred dynamically, and the running time is (roughly) governed by the number of terms in the output. In previous versions of `SF`, an instance of `tos` could be accelerated by optionally providing a partition list containing the support of the output; now, this option is ignored.
- The orthogonal bases created by `add_basis` are now defined in terms of triangularity with respect to the conjugate-lexicographic order $<_C$, not the lexicographic order $<_L$

(see the definitions in §5). This has no effect on computations involving the orthogonal bases discussed in §4, but has the benefit that in all cases, the basis elements indexed by partitions with $\leq n$ rows may be specialized to a basis for the ring of symmetric polynomials in n variables.

- An optional flag may be provided to `add_basis`, indicating that the basis being created is known to be triangular with respect to all refinements of the dominance order. This information is used to accelerate the Gram-Schmidt algorithm.
- The '`to`'-procedures created by `add_basis` are now internally and externally similar to the new `tos`. In particular, they now derive the support of the output dynamically, and ignore any user-provided information about this support. Also, they now allow the option of computing in the ring of symmetric polynomials in n variables (i.e., modulo the ideal generated by $e_{n+1}, e_{n+2}, \ldots$).
- The procedures for converting to and from a basis created via `dual_basis` have been redesigned. The old versions consumed large amounts of memory, and large instances could trigger crashes in old versions of Maple. The new versions are much more space-efficient, do not trigger crashes, and in most cases, are faster.
- The procedure `dominate` may now be used to test whether two partitions are related in dominance order.
- The procedure `nextPar` has been added to the package.
- The output of `evalsf` is now filtered through `normal` in all cases, even if the second argument is a symmetric function of degree 0.
- Most of the examples have been rewritten and are now faster and more space-efficient. Also, `seminormal` now produces matrix entries with smaller denominators.

Version 2.3. In addition to the debut of the Vanilla Edition, the significant changes and new features in `SF` are:

- Two new examples, `jacks` and `kfpoly` have been added.
- The example `m_conjecture` has been deleted, on grounds of obsolescence [**3**].
- Nearly all of the examples have been rewritten, with improved documentation, and (in some cases) minor changes in functionality.
- A bug in `tos` and `to`-procedures defined by `add_basis` involving misuse of Maple's "vector degree" has been fixed. This bug could appear only in rare situations and would manifest itself by producing a Maple `ERROR`. Many thanks to Mike Zabrocki ⟨`zabrocki@math.uqam.ca`⟩ for spotting the bug.
- A minor bug in `dominate` has been fixed. When asked to list partitions with at most n parts dominated by a partition with more than n parts, it now (correctly) returns an empty list.
- `stdeg` has been restored. It had been downgraded to an internal role in Version 2.0.

Version 2.2. Aside from cosmetic improvements, the significant changes and new features in this version of `SF` are:

- The procedure `evalsf` has been enhanced. It is still upward-compatible with the old version, but is now capable of plethystic evaluations. See the help text.
- To minimize the chance of collision with user-defined names, the global variable `Bases` has been renamed `` `SF/Bases` ``.

- The example `skew_shapes` has been added, and the example `graphs` has been renamed `graph`.

Version 2.1. There were no changes in functionality.

Version 2.0. The main changes affecting users upgrading from Versions 1.x are:
- Every basis-conversion procedure, such as `toe`, `toh`, `top` and `tos` now returns as output an expression that is collected in 'distributed' form (see the Maple documentation for `collect`), with `normal` applied to the coefficients.
- Every procedure in `SF` that accepts symmetric functions is now able to correctly process non-homogeneous symmetric functions.
- The code for manipulating monomial symmetric functions (and in particular, `tom`) is no longer built-in. It has been replaced by the more general `dual_basis` construction.
- `ch2p` and `p2ch` have been replaced with `char2sf` and `sf2char`. The new procedures allow conversions between symmetric group characters and arbitrary symmetric functions, not just expressions in the power-sums.
- `ibocaj` and `jacobi` have been replaced by `jt_matrix`.
- `Par` no longer uses `option remember`.
- `stdeg` is now an internal procedure, renamed `` `SF/stdeg` ``.

8. Miscellany

Global Warning. The following names have been reserved for use by `SF` and should not be assigned values by the user:

1. `e,h,p,s` (names of predefined bases).
2. `e1,h1,p1,e2,h2,p2,...` (names of symmetric functions).
3. `cl` (the name reserved for class functions on the symmetric group).

Maple Variations. A new release of Maple often means fundamental changes in the underlying syntax. For example, the ditto operator changed from `"` to `%` in Maple V Release 5, and the concatenation operator changed from `.` to `||` in Maple 6. Although the source code for `SF` has been completely updated to account for these changes, the reader should beware that there are a few instances in the documentation of individual functions where usage is explained in terms of the old syntax.

Object too large. Old versions of Maple (prior to Maple 6) have noticeable limits on the number of operands in a Maple object. On a 32-bit x86 CPU, computations involving generic symmetric functions of degree 35 will trigger an '`object too large`' error in these versions of Maple. Later versions do not have these limitations.

Under the hood. I am frequently asked about the algorithm `SF` uses to convert a symmetric function f into Schur functions. The basic idea is that there is a total ordering of the set of h-monomials such that h_λ is the "first" term of the Jacobi-Trudi h-expansion of s_λ. Thus we first convert f into an h-polynomial, and then extract the leading h-monomial from f, say h_λ. The coefficient of h_λ in f is also the coefficient of s_λ in f; we then expand s_λ into an h-polynomial via (say) the Jacobi-Trudi determinant, subtract the appropriate multiple of the h-expression for s_λ from f so as to kill the leading term, and iterate.

Notice that the main expense of this algorithm is the cost of converting f and each s_λ to the h-basis. The running time is strongly correlated with the number of distinct Schur functions in the output, and (unlike the implementations of `tos` in older versions of `SF`) there is no advantage in knowing the support of this output in advance.

The '`to`'-procedures created by `add_basis` operate in a similar way.

More Hints. There is one case in which it is possible to use knowledge of the support of the output to improve the performance of a '`to`'-procedure. Indeed, suppose that one intends to compute the `B`-expansion of a symmetric function `f`, where `B` is a basis that was created by `dual_basis`. If one can build in advance a list that includes every partition in the `B`-support of `f`, then the function call `toB(f,`*partition list*`)` will be able to do the conversion by computing scalar products involving only the elements in the dual basis indexed by the given list, rather than involving all members of the dual basis.

For example, the coefficient of m_μ in s_λ is nonzero only if λ dominates μ (see (2)), so `dominate([4,3,2,2])` is a partition list that contains the `m`-support of `s[4,3,2,2]`. As an experiment, try running the commands

```
with(SF); dual_basis(m,h);
parlist:=dominate([4,3,2,2]);
tom(s[4,3,2,2], parlist);
tom(s[4,3,2,2]);
```

We find that the first call to `tom` runs about 3 or 4 times faster than the second.

Bugs. Please send reports of reproducible bugs to my e-mail address on page 1. Be sure to include an example, preferably as small as possible. A common problem that is not a bug occurs when the user assigns values to the special names `e`, `h`, `p`, `s`, or `m` (if `m` is the name of a basis added during an `SF` session). I've made this mistake myself on more than one occasion.

Compensation. If `SF` proves to be useful in your own research, I would appreciate an acknowledgment of it in publications derived from that research.

Mailing list. I maintain a low-traffic, private mailing list for users of `SF` and my other packages `posets`, `coxeter`, and `weyl`. If you would like to be kept informed of new versions, new packages, or (gasp) bugs, please send me an e-mail message.

9. Copyleft

10. References

[**1**] W. Fulton and J. Harris, "Representation Theory. A First Course," Springer-Verlag, Berlin-New York, 1991.

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

[**3**] M. Haiman, Hilbert schemes, polygraphs and the Macdonald positivity conjecture, *J. Amer. Math. Soc.* **14** (2001), 941–1006.

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

[**5**] G. D. James and A. Kerber, "The Representation Theory of the Symmetric Group," Addison-Wesley, Reading, MA, 1981.

[**6**] D. E. Littlewood, "The Theory of Group Characters," 2nd ed., Oxford Univ. Press, Oxford, 1950.

[**7**] I. G. Macdonald, "Symmetric Functions and Hall Polynomials," 2nd ed., Oxford Univ. Press, Oxford, 1995.

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

[**9**] B. E. Sagan, "The Symmetric Group. Representations, Combinatorial Algorithms, and Symmetric Functions," 2nd ed., Springer, New York, 2001.

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

[**11**] R. P. Stanley, "Enumerative Combinatorics, Vol. 2," Cambridge Univ. Press, Cambridge, 1999.

[**12**] A. Takemura, "Zonal Polynomials," Institute of Mathematical Statistics, Hayward, CA, 1984.