Theor Chim Acta (1991) 79: 1–42

Theoretica Chimica Acta

The automated solution of second quantization equations with applications to the coupled cluster approach*

Curtis L. Janssen and Henry F. Schaefer III

Center for Computational Quantum Chemistry, University of Georgia, Athens, GA 30602, USA

Received August 14, 1990; received in revised form/Accepted September 26, 1990

Summary. Theoretical methods in chemistry frequently involve the tedious solution of complex algebraic equations. Then the solutions, sometimes still quite complex, are usually hand-coded by a programmer into an efficient computer language. During this procedure it is all too easy to make an error which will go undetected. A better approach would be to introduce the computer at an even earlier stage in the development of the theory by programming it to first solve the set of equations and then compile the solution into an efficient computer language. In this research a program has been written in the C programming language which can efficiently compute the quasivacuum expectation value of a product of creation and annihilation operators and scalar arrays. The terms in the resulting expressions are then transformed into a canonical form so that all equivalent terms can be combined. Finally, the equations are compiled into a simple representation which can be rapidly interpreted by a Fortran program. This symbol manipulator has been applied to open-shell coupled cluster theory. Two coupled cluster methods using high-spin open-shell references are presented. In one of these methods, the cluster operator contains the unitary group generators, and products thereof, which generate all single and double excitations with respect to the reference. The other uses a simplified cluster operator which generates equations that must be spin-projected. These methods are compared to other descriptions of electron correlation for the CH_2 singlet-triplet splitting and the NH_2 potential energy surface.

Key words: Second quantization equations – Coupled cluster approach – Electronic structure – Schrödinger equation – C programming language

1. Introduction

Theoretical methods in physical science frequently involve the tedious solution of complex algebraic equations. To aid researchers in deriving such equations,

* A condensation of the Ph.D. thesis of Curtis L. Janssen, Department of Chemistry, University of California, Berkeley, California, March 1990

elaborate methods have been devised in which the equations can be represented as diagrams [1–4]. These diagrammatic approaches are very powerful and greatly reduce the amount of labor needed to obtain the necessary equations. However much these techniques reduce the researcher's chore, the complexity of the equations which can be derived remains constrained by a human's ability to manipulate the equations, whether by diagrammatic or any other means.

This constraint would not pose any problem if human imagination was so limited that the only theoretical approaches to the description of nature which could be conceived had a simple solution. Fortunately, this is not the case. Unfortunately, very complex equations can be generated by even conceptually simple physical theories.

Similarly, even after a simple set of equations has been obtained, their evaluation can be very difficult if done by hand. To be more concrete, the self-consistent eigenvalue problem which arises in Hartree–Fock theory for molecular orbitals composed of a linear combination of atomic orbitals [5] can be easily written on a few pages of paper, but only the simplest physical systems can be studied without the use of an electronic computer. With the advent of these machines the use of Hartree–Fock theory and corrections thereof to the study of chemical problems became a field in itself. There is no reason that the same computers which have been found to be essential in the evaluation of the equations resulting from physical theories cannot be used to derive the equations pertinent to unexplored theories as well.

In the present work, we are concerned with electronic structure in chemical systems. More specifically, we are investigating the solution of the time independent Schrödinger equation for electrons moving in a nuclear Coulomb field where the Born–Oppenheimer approximation is invoked and, hence, the nuclei are considered to be stationary. The expansion of the wavefunction in terms of one particle functions has been extensively used and is in excellent starting point for a theory describing electron correlation. This starting point allows the second quantization formalism to be used as a convenient method for describing the behavior of electrons in terms of the one particle functions. It is the basis for many algebraic and diagrammatic approaches to electron correlation. We have developed a computer program for deriving equations which can be expressed in this formalism. The program allows algebraically complex expressions to be derived and writes these equations in the simplest possible final form. This program, referred to as the second quantization symbol manipulator, SQSYM, will be described in detail in Sect. 2.

The ability to obtain solutions to theories which suffer a very complex algebraic structure would be of little use if we could not apply the solutions to specific chemical systems. That is, it is essential that the solutions be expressed in some computer programming language, such as Fortran. If SQSYM accomplishes its stated goal, then it will be able to produce equations which are too complex for a human to code in a programming language. So, yet again, one turns to the electronic computer, but, this time, for an automated compilation of the equations into a high-level programming language. This is implemented as an extension to SQSYM and is also detailed in Sect. 2.

The automated solution and compilation of physical theories was never a goal in itself. Our principal interest has always been electronic structure theory. We found the range of theories which could be investigated limited by the time consuming process of doing tedious algebraic or diagrammatic manipulations. The area we found most interesting, open-shell coupled cluster theory, was so algebraically complex that the project seemed hopeless if one had to do the derivations by hand. Thus, SQSYM was born. We found it necessary to discuss the details of SQSYM in the present work, since its development became a research project in its own right, but the essence of the research is found in the final sections. There are described the versions of open-shell coupled cluster theory for which SQSYM was used to transform from a concept into a method which could produce physical observables for general chemical systems.

Before the open-shell coupled cluster theories are discussed, motivation for pursuing these algebraically complex methods should be provided. The basic notion of coupled cluster theory is that the wavefunction of a many body system can be well represented in terms of only few body interactions. This idea also appears in the virial expansion for an imperfect gas, where the virial coefficients can be broken down into one, two, three, etc. body contributions. For dilute gases, the one and two body contributions provide a good approximation to the virial expansion [6]. Moving to the quantum mechanical regime, the method of self-consistent fields is usually the starting point for the problem of many interacting particles. This method reduces the many body problem to several one body problems. In the coupled cluster approach [4, 7] the self-consistent field wavefunction is improved upon by writing the exact wavefunction as

$$\Psi = e^{T}\Phi,$$
$$T = T_1 + T_2 + T_3 + \cdots,$$

where Ψ is the exact wavefunction, Φ is the self-consistent field wavefunction, T_1 represents one particle excitations, T_2 represents two particle excitations, and so on. The operator T is frequently called the cluster operator and e^{T} is referred to as the wave operator. Letting N be the number of particles, this expansion is exact if all T_i up to T_N are included. To reduce the complexity of the equations a common approximation is to let $T = T_1 + T_2$. This approximation is exact for two particles and even exact for certain many particle systems. For example, if we have $N/2$ noninteracting systems, each with two electrons, and labeled A, B, C, etc., then

$$\begin{aligned}\Psi_{\mathrm{AB}\cdots} &= e^{T_{\mathrm{A}}}\Phi_{\mathrm{A}} e^{T_{\mathrm{B}}}\Phi_{\mathrm{B}} \cdots \\ &= e^{T_{\mathrm{A}} + T_{\mathrm{B}} + \cdots}\Phi_{\mathrm{AB}\cdots}.\end{aligned}$$

Thus, for this N body system, $T = T_1 + T_2$ generates the exact wavefunction. The question is how will this theory perform when these two body systems are allowed to interact.

If we look back to the example of the dilute imperfect gas for an indication of the quality that can be expected by including up to only two body interactions in our many electron theory, we might expect that it will fare quite well. But a fundamental difference between the electronic many body problem and the imperfect gas is that while a dilute gas would have almost exclusively two body collisions, a given electron in a molecule will feel the repulsion of many other electrons simultaneously. That is, the Coulomb force acts over a much longer range than the interactions between gas molecules, relative to the mean distance between particles. Despite this, the self-consistent field (SCF) approximation turns out to be a fairly good description of molecules because the Coulomb force felt by an electron can be replaced to a large extent by the average force due to all other electrons. The short range of the fluctuation potential in atomic systems [8] demonstrates this and the many body terms can be sensibly approximated by products of independent few body terms, given that an SCF wavefunction is used as the starting point.

However, even the SCF wavefunction can be expressed as the exponential of one body operators by using Thouless's theorem [9], $\Phi = e^{T_1}\Xi$, where Ξ is an antisymmetrized product of atomic orbitals. This fact explains the success of coupled cluster methods which set $\Psi = e^{T_2}\Phi$. This coupled cluster doubles (CCD) wavefunction contains, for instance, quadruple excitations of electrons, but the coefficients of these states are approximated as the products of the coefficients of doubly excited states. The advantage of representing the quadruple excitations in this way is that the number of unknowns which must be determined for this theory is much smaller than the number needed to exactly represent all the coefficients of all excited states. So from an intuitive viewpoint the coupled cluster approach seems very appealing. However, the ultimate test is how well experimental results can be reproduced, or, better yet, predicted. There is a growing body of evidence [10–13] suggesting that coupled cluster theory, at least for closed-shell systems, is a better approximation than methods of similar computational complexity, for example the configuration interaction (CI) approach.

However, for open-shell systems the problem is not yet completely solved, despite the fact that a great deal of work has been invested into the problem of generalizing the reference state [14]. Most of this effort has been directed towards the development of multireference coupled cluster theories. Instead of using a single Slater determinant as a reference function, these methods utilize wave operators which act upon a reference function formed from a linear combination of Slater determinants. The space of determinants which is used to form the reference is known as the model space. The selection of model space is used to classify the orbitals. Typically the orbitals occupied in all model functions are referred to as holes, those which are occupied in some model functions are valence orbitals, and orbitals which are occupied in no model space functions are particle orbitals. The number of electrons in the valence orbitals will be called N_V. Commonly used model spaces are the complete and quasicomplete model space. The complete model space contains the functions corresponding to all possible occupations of the valence orbitals with N_V electrons. The quasicomplete model space further partitions the valence orbitals into subgroups. The number of electrons in each of these subgroups is fixed, and all functions which have the appropriate number of electrons in each subgroup are included in this model space. It would be most desirable to have a method which can work with a general incomplete model space, but we must consider how the selection of the model space affects the connectedness of the expressions for the effective Hamiltonian and the size-extensivity of the effective Hamiltonian's roots. While it is now fairly well understood that size extensive energies may be obtained for the compelte model space, some questions still remain about more general model spaces [15]. The multireference formalisms tend to be considerably more complex than their single reference counterparts. Moreover, the methods are computationally expensive and in some cases near singularities arise in the equations to be solved.

One group of multireference coupled cluster methods is based on the technique of Jeziorski and Monkhorst [16]. In 1981 Jeziorski and Monkhorst proposed a multireference coupled cluster method in which the exact wavefunction is expressed as a linear combination of cluster expansions:

$$\Psi_\nu = \sum_{\mu}^{n} c_{\nu\mu} e^{T^\mu} \Phi_\mu,$$

where the linearly independent set of n functions, Φ_μ, form the model space and the $c_{\nu\mu}$ are obtained by the diagonalization of an effective Hamiltonian which has dimension n to yield roots, Ψ_ν, $1 \leqslant \nu \leqslant n$. The connectedness of this approach for the complete model space was demonstrated by Jeziorski and Monkhorst. Laidig and Bartlett [17] implemented a simplified version of this method in 1984. Their method included only terms linear in the cluster operator and was restricted to the ground state of the system of interest. Laidig et al. [18] went on to use a spin-adapted formalism in 1987. This approach uses the graphical unitary group approach to generate the requisite matrix elements. The method of Jeziorski and Monkhorst was also implemented by Jeziorski and Paldus [19] in 1988. In this case the only approximation was the restriction to single and double excitations in the cluster operator and the exclusion of terms which were nonlinear in the cluster operator. These authors used a spin-adapted formalism and restricted the model space to the specific but important case of a complete space in two orbitals of different symmetry. For this choice of model space there are two model functions both of which are closed-shell wavefunctions. The restriction to linear terms was upgraded to a restriction to quadratic terms by Paldus et al. [20]. They found that, like in the single reference linearized coupled cluster case, the quadratic terms help eliminate the singularities that can arise in the linearized theories. Another implementation of the Jeziorski and Monkhorst method was accomplished by Meissner et al. [21] in 1988. They restricted the cluster operator to two particle excitations only, but fully treated the nonlinear terms. The model space had to be of the complete variety, but in 1989 Meissner et al. [22] extended the method to include special classes of incomplete model spaces while still achieving size extensive expressions for the energy and Meissner and Bartlett [23] went on to develop an approach for general incomplete model spaces in 1990.

Spin adaptation of multireference coupled cluster theory is rather straightforward for linearized methods, but is quite difficult when nonlinear products of the cluster operator are kept. The only example of a spin-adapted multireference coupled cluster theory that retains nonlinear terms discussed thus far is that of Paldus et al. [20]. Banerjee and Simons [24] set out to develop a spin-adapted multireference coupled cluster method in 1981. Their cluster operator, $\boldsymbol{T}$, was expressed in terms of the generators of the unitary group:

$$\boldsymbol{E}^p_q = \sum_{\sigma}^{\alpha\beta} \boldsymbol{a}^\dagger_{p\sigma}\boldsymbol{a}_{q\sigma}.$$

Unitary group generators are used to ensure that $[\boldsymbol{S}^2, \boldsymbol{T}] = 0$. With this form for $\boldsymbol{T}$, the operator $\boldsymbol{T}$ and its powers do not introduce spin contamination into the wavefunction. The prescription they present involves first obtaining a complete active space (CAS) SCF wavefunction, $|\Psi\rangle$. This wavefunction is to be frozen throughout the rest of the procedure. Then, they approximate $\boldsymbol{T}$ as $\boldsymbol{T}_1 + \boldsymbol{T}_2$, where

$$\boldsymbol{T}_1 = \sum_{a,x} t^x_a \boldsymbol{E}^a_x,$$

$$\boldsymbol{T}_2 = \sum_{a \leqslant b,x,y} t^{xy}_{ab} \boldsymbol{E}^a_x \boldsymbol{E}^b_y.$$

Here and elsewhere labeling conventions will be used to specify the ranges of summations. The indices i and j will refer to holes, x and y will label valence orbitals, a and b will designate particles, and p, q, r, and s label indices which can take on any value. With this choice of T, Banerjee and Simons were able to obtain and apply a theory including in the energy terms up to quadratic in T. Baker and Robb [25] implemented a theory similar to this, but used only $T = T_2$ and modified it to be anti-Hermitian. Then they minimized the energy:

$$E = \langle \Psi | \{e^{T_2}\}^{\dagger} H e^{T_2} | \Psi \rangle = \langle \Psi | e^{-T_2} H e^{T_2} | \Psi \rangle.$$

This energy expansion was truncated to quadratic terms in T. Thus, when the energy is minimized with respect to variations in the cluster amplitudes, the equations obtained for the amplitudes were linear in T. It should be noted that the energy expression which is being minimized is an approximate energy expression and thus the computed energy may come out below the full CI energy, which is the exact solution within the given basis set. The procedure of Banerjee and Simons and that of Baker and Robb suffer severe problems in their practicability. The size of the CASSCF expansion grows exponentially with the number of valence orbitals for a given number of electrons. Since this greatly limits the size of the valence space, it is desirable to include core electron correlations through the cluster operator. However, the above methods do not possess this ability. In 1988, Hoffmann and Simons [26] resolved this problem with their unitary coupled cluster (UCC) method. An anti-Hermitian cluster operator was used to obtain a unitary wave operator and the resulting energy expression was truncated at quadratic terms in T and minimized with respect to variations in T. All single and double excitations were included in their cluster operator. This method is computationally expensive; however, Hoffmann and Simons went on to present [27] a related, but computationally efficient formalism, UCEPA, one year later.

Another important group of multireference methods, whose overview shall be left to Mukherjee and Pal [14], is formulated around a single wave operator which generates not only the $\Psi_{\nu}^{(N_V)}$, but also the ionized system wavefunctions $\Psi_{\nu}^{(n)}$, where $0 \leq n < N_V$. This introduces a surplus of unknowns and equations if one is interested in only unionized states of a chemical system, as is frequently the case. These methods receive strong support because they transparently remove linear dependencies between the cluster amplitudes and also because it is better understood that the energies produced by these methods are size extensive for incomplete model spaces.

Despite the progress in the multireference coupled cluster methods, the investigation of single reference techniques for open-shell systems has not yet been exhausted. The role which these methods would fill is the efficient and accurate prediction of properties for open-shell systems which are dominated by one reference. The reference could be either a single Slater determinant or a simple linear combination of determinants where the coefficient of each determinant is constrained by the desire to obtain a proper eigenfunction of S^2. The coupled cluster formalisms based on unrestricted spin orbitals can already be used to treat the high-spin open-shell case, where the number of α spin occupied orbitals is greater than the number of β spin occupied orbitals. This method suffers from a spin contaminated wavefunction and energy; that is, the wavefunction is not an eigenfunction of the S^2 operator and contributions from the contaminants appear in the energy expression. These methods also require that more unknowns are present than are actually required to solve the problem. Rittby and Bartlett [28] pointed out that if an unrestricted Hartree–Fock (UHF) reference is discarded in favor of a restricted Hartree–Fock (RHF) reference, then, even in a spin orbital (as opposed to spin eigenfunction) formulation, some relief to these difficulties can be found. In this case, spin contaminated contributions to the energy disappear, while the wavefunction itself retains contributions of the wrong spin symmetry. They refer to this method as projected coupled cluster (PCC). Projected coupled cluster still requires determining more unknowns than should be necessary.

Although differing somewhat from the traditional coupled cluster theories, Nakatsuji and Hirao suggest a method [29, 30] for the closed-shell reference and high-spin open-shell reference case, which has been designated symmetry adapted cluster (SAC) theory, where the cluster operator generates only those excitations from the reference which are of the appropriate spin symmetry. Nakatsuji and Hirao truncated their cluster operator to only single excitations in all of the systems with open-shell references that they studied using this method, although some of these did not interact with the reference through one body operators. This cluster operator does not necessarily generate spin eigenfunctions when it operates on functions other than the reference; thus, the wave operator generates a wavefunction which is not an eigenfunction of spin. They chose to deal with this by projecting out the spin contaminants. All of these contaminants come from the terms which are nonlinear in the cluster operator. Hirao and Nakatsuji treated higher excitations from an open-shell reference with another method [31], where open-shell singlet and triplet wavefunctions were obtained by starting with the reference determinant

$$\Phi = |\phi_1\alpha\phi_1\beta\phi_2\alpha\phi_2\beta \cdots \phi_m\alpha\phi_n\beta\rangle,$$

where ϕ_i are the spatial orbitals obtained from a RHF procedure. The cluster operator was chosen to include all interacting single and double excitations from the reference. To obtain an expression for the energy or equations for the cluster coefficients the heavily spin contaminated wavefunction $\Psi = e^T\Phi$ is operated upon by a singlet or triplet spin projector to remove the undesired components. A method for describing open-shell singlets and triplets without starting from such a poor reference would be better.

In the present work the symbol manipulator, which will be discussed in Sect. 2, will be used to assist in the development of several coupled cluster methods for describing electron correlation in systems that are well approximated by a single high-spin open-shell reference determinant. Section 3 will begin by outlining the basic approach by applications to a method which is already well understood, specifically, configuration interaction. These and all other applications of the symbol manipulator will include all single and double excitations which directly interact with the reference. Section 3 continues with a spin-adapted open-shell coupled cluster method truncated to cubic terms in the cluster operator and then ends with an approach which bears resemblances to the SAC method of Nakatsuji. Section 4 will present results for a few chemical systems and Sect. 5 closes with a discussion of the main results.

2. The automated solution of second quantization equations

Many theoretical approaches can be described in the second quantization formalism. Perturbation theory, configuration interaction, coupled cluster, and other methods which use one particle orbitals as their starting point have realizations in this formalism. We shall focus our attention on coupled cluster with the standard nonrelativistic electronic Hamiltonian. After the form of the reference function, $|\Phi\rangle$, and the cluster operator $\boldsymbol{T}$ have been decided upon, all that remains is to evaluate the expression

$$E_{cc} = \langle\Phi|e^{-T}He^{T}|\Phi\rangle, \tag{2.1}$$

using projections upon excited states, $|X\rangle$, to determine the $\boldsymbol{T}$ operator:

$$\langle X|e^{-T}He^{T}|\Phi\rangle = 0. \tag{2.2}$$

A second quantization representation of $\boldsymbol{T}$ and $\boldsymbol{H}$ leads to a very straightforward solution of the above equations. While the manipulations required to obtain the solution do not require much innovation, it does require a great deal of error-prone work. This complexity motivated diagrammatic approaches, without which, much of the published work in perturbation theory, coupled cluster, etc. could not have been done. But for a sufficiently complex choice of $\boldsymbol{T}$, even the diagrammatic methods can become unwieldy and subject to a multiplicity of human errors. This is the motivation for an automated solution to this problem.

The computer program developed in this research for obtaining the solutions to these equations will be called the second quantization symbol manipulator (SQSYM). This program understands simple commands which are used to describe the operators and then manipulate the expressions. The expressions produced can be very complex. In some cases rearranging the expressions in such a way that they can be efficiently executed on a computer, allowing the energy actually to be computed for some arbitrary molecule, would be a monumental task, perhaps even impossible if the task was left to human hands alone. Thus, after the expressions have been obtained and rewritten in their simplest form, SQSYM must be able to take the expressions and compile them into a language which can be efficiently executed.

This section is concerned with describing the approach taken by SQSYM to solve these problems. First, the second quantization formalism will be reviewed, then the language which the SQSYM interpreter understands will be outlined. Following this, we will discuss the implementation details for SQSYM. This entails such things as the data representation and data manipulation.

2.1. Second quantization formalism

The second quantization formalism [32, 33] provides a simple method for computing matrix elements of operators between states written as an antisymmetrized product of orbitals, each with single occupancy. These states are built up from the true vacuum, $|\rangle$, by the particle creation operators, $a_p^\dagger$:

$$|\Phi\rangle = \prod_p a_p^\dagger |\rangle. \tag{2.3}$$

The adjoint of a creation operator is a particle annihilation operator, a_p. To create properly antisymmetrized wavefunctions the creation and annihilation operators must obey the anticommutation relations

$$\{a_p^\dagger, a_q\} = \delta_{p,q}, \tag{2.4a}$$

$$\{a_p^\dagger, a_q^\dagger\} = 0, \tag{2.4b}$$

and

$$\{a_p, a_q\} = 0. \tag{2.4c}$$

Operators can be expressed as sums of products of creation and annihilation operators. The one body operators can be expressed as

$$O_1 = \sum_{p,q} \langle p|o_1|q\rangle a_p^\dagger a_q \tag{2.5}$$

and two body operators can be written

$$O_2 = \tfrac{1}{2} \sum_{p,q,r,s} \langle pq|o_2|sr\rangle a_p^\dagger a_q^\dagger a_r a_s, \tag{2.6}$$

where

$$\langle p|o_1|q\rangle = \int dr \phi_p(r) o_1 \phi_q(r) \tag{2.7}$$

and

$$\langle pq|o_2|rs\rangle = \int dr_1\, dr_2 \phi_p(r_1)\phi_q(r_2) o_2 \phi_r(r_1)\phi_s(r_2). \tag{2.8}$$

For the case of two electron integrals the notation

$$(pr\,|\,qs) = \langle pq|\frac{1}{r_{12}}|rs\rangle \tag{2.9}$$

is frequently used. The summations in these general operators must run over all the basis functions.

It is inconvenient to have to deal with the product of large numbers of creation operators which occur in Eq. (2.3). It is better to redefine the vacuum to be some reference function,

$$|0\rangle = \prod_i a_i^\dagger |\rangle, \tag{2.10}$$

following which all of the states we must deal with can be expressed in terms of the quasivacuum, $|0\rangle$, by applying the desired electron creation and annihilation operators. For example, a single excitation from the reference can be written as creation of a hole (annihilation of an orbital that is occupied in $|0\rangle$) followed by creation of a particle (creation of an occupied orbital which was previously unoccupied in $|0\rangle$). This state would be written $a_a^\dagger a_i|0\rangle$.

Now we have all of the tools that are necessary to evaluate matrix elements with this formalism. Any matrix element can be written as the quasivacuum expectation value of a product of creation and annihilation operators contracted with the appropriate matrix elements. Observing that any given creation or annihilation operator, X, will satisfy either $X|0\rangle = 0$ or $\langle 0|X = 0$, we can use Eq. (2.4) to commute the creation and annihilation operators around each other until they act upon either the quasivacuum or its adjoint to give zero. After all of the creation and annihilation operators have been eliminated in this manner a strictly scalar expression will remain. This is the desired matrix element. This process can be simplified by using Wick's theorem [34], which implicitly performs all of the required commutations for us. Wick's theorem can be written

$$\begin{aligned}
ABCD\cdots XYZ = {}&:ABCD\cdots XYZ:\\
&+:\overline{AB}CD\cdots XYZ:+:\overline{ABC}D\cdots XYZ:+\cdots+:ABCD\cdots X\overline{YZ}:\\
&+:\overline{A\overline{BC}D}\cdots XYZ:+:\overline{A\overline{BC}D}\cdots XYZ:+\cdots+:ABCD\cdots \overline{XYZ}:\\
&+\cdots\\
&+:\overline{AB}\,\overline{CD}\cdots \overline{X}\,\overline{YZ}:+:\overline{A\overline{BC}D}\cdots \overline{X}\,\overline{YZ}:+\cdots+:\overline{ABCD}\cdots \overline{XYZ}:
\end{aligned}\tag{2.11}$$

or

an operator = the normal ordered operator + all possible normal ordered products with single contractions + all possible normal ordered products with two contractions + · · · + the fully contracted products, where the normal ordering of operators is represented by placing colons on either side of the group of operators to be normal ordered. The normal ordering process moves operators which annihilate $\langle 0|$ to the left or, equivalently moves operators which annihilate $|0\rangle$ to the right. This motion is done with the sign modifications specified by the anticommutation relations, but the contractions which come from the delta functions in the anticommutation relations are ignored.

To relate an operator to its normal ordered form, the contractions which arise in the anticommutation process must be considered and are represented by a line drawn between two operators. A contraction generates a Kronecker delta between the indices of the operators being contracted. If the operators are in different spaces (one operates in the particle space and one operates in the hole space) the delta function is zero. Similarly, if the operators do not need to be commuted to obtain a normal ordered form, then a contraction between these two operators must not appear. Also, if the two operators are of the same type (both are creation operators or both are hole operators), then the contracted term does not appear. Finally, the sign of each term must be adjusted to reflect the number of commutations which had to be performed to get that term. This can be simply computed by taking the sign change needed to normal order the operators remaining after the contractions have been performed multiplied by $(-1)^{n_{\text{crossings}}}$, where $n_{\text{crossings}}$ is the number of times the contraction lines in Eq. (2.11) cross.

When expectation values with respect to the reference wavefunction are computed using Wick's theorem, the only terms to survive are the fully contracted products.

Due to the complexity of the above procedure elegant diagrammatic methods have been developed to simplify the derivation process. With these methods one represents matrix elements and the cluster operator as points on a page. Then, one draws rays corresponding to creation and annihilation operators entering or leaving these points. The rays between matrix elements and operators are then connected in all possible ways to obtain diagrams which are related to the terms which the anticommutation relations could give in a slightly more straightforward yet much more tedious manner.

The indices of the creation and annihilation operators used so far in this section specify spin orbitals. That is, the product of a spatial one electron orbital, ϕ_p, and an electron spin function, α or β. In much of what follows we shall require that each spatial orbital, ϕ_p, is used with two spin orbitals, $\phi_{q\alpha}$ and $\phi_{p\beta}$. This is the starting point for building wavefunctions of the correct electron spin symmetry and the use of the same spatial orbital for two spin orbitals is known as the restricted Hartree–Fock (RHF) procedure.

Now it is necessary to associate with each creation and annihilation operator two indices, one for the spatial part and one for the spin part. So we have $\boldsymbol{a}_{p\sigma}$, where p refers to the spatial orbital and σ takes on the values α or β and refers to the spin. From these operators we can define the particle number and spin conserving operators

$$\boldsymbol{E}^p_q = \sum_{\sigma}^{\alpha\beta} \boldsymbol{a}^\dagger_{p\sigma}\boldsymbol{a}_{q\sigma}, \tag{2.12}$$

which are known as the generators of the unitary group.

Since the Hamiltonian is both particle number and spin conserving it can be expressed in terms of the unitary group generators:

$$\boldsymbol{H} = h_{pq}\boldsymbol{E}^p_q + \tfrac{1}{2}(pq\,|\,rs)(\boldsymbol{E}^p_q\boldsymbol{E}^r_s - \delta_{qr}\boldsymbol{E}^p_s), \tag{2.13}$$

where repeated indices imply summations. Similar expressions exist for other operators written with the second quantization formalism.

2.2. The SQSYM interpreter

SQSYM is an interpreter. That is, it reads input from a file or terminal and executes the commands as they are read. This is in contrast to a compiler, where the commands to be executed are first translated into another language. Frequently, this language is the machine's most primitive and efficient language. There is always extra overhead associated with interpreting the commands instead of running a program that has been compiled into a very efficient language. However, for SQSYM, this overhead is very small. The SQSYM language is designed to allow much to be done with very few commands. For example, there is one command that takes the product of several expressions and then computes the expectation value of the resulting expression. The vast majority of the time is spent performing this complex task, while very little time is needed to determine what command has been requested, where the data are located, and what routines must be called.

SQSYM can also act as a compiler. However, it does not compile its input program into a more efficient language like most other compilers. Instead, the input is always interpreted with the result of producing data that are internal to SQSYM and which represent mathematical expressions. When the compiler portion of SQSYM is invoked, these expressions are compiled into a form which can be executed. This target language consists of primitive commands that perform various operations between one or two arrays. To obtain an energy for some molecular system the compiler output must be interpreted by another program.

There are two different routes which could have been followed in implementing SQSYM. A previously existing symbol manipulator such as Macsyma, Reduce, or Mathematica [35] could have been used as a starting point. All of these packages are large and complex; one person could not reproduce all of the effort that went into these packages. However, most of the features of these commercial programs are not needed for the present purposes. Such things as analytic integration are not necessary in SQSYM. Furthermore, each of these packages has severe deficiencies with regard to manipulation of second quantization equations. Although Reduce has been used successfully for simple second quantization problems in quantum chemistry [36], much customization of Macsyma, Reduce or Mathematica would be required to efficiently derive and simplify the large and complex expressions which must be dealt with in the present work. Furthermore, they would be useless for the conversion of the equations to forms which could be either compiled or efficiently executed. Macsyma and Mathematica do have primitive routines which can produce Fortran code segments; however, this feature would not be able to produce efficient code where multidimensional arrays are concerned. Moreover, the use of such commercial products may restrict the number of machines which SQSYM could run on. The task at hand is sufficiently computationally intensive that it demands that the fastest machine available at any given time be used to run the application. Thus, the idea of starting from an existing commercial product had to be abandoned.

Ruling out commercial symbol manipulation packages as a starting point does not complete the choice of implementation, however. Next the language in which SQSYM is written must be chosen. Three possibilities were considered, Fortran, Lisp [37], and C [38]. Fortran is a widely available and efficient language, but, it is not a good choice for large complex programs. The limitation of six characters on the lengths of symbols such as variable names and subroutine names make it difficult to give meaningful names in all cases. When hundreds of subroutines and as many variables are being used, the ability to assign descriptive names speeds the programming and debugging processes. Another problem with Fortran is the limited number of data types that are available. Fortran provides for integers, finite precision real numbers, characters, and arrays of these data types. It is much more convenient to be able to define user data types, so that pieces of related data can be moved around as one unit. The data types SQSYM uses are somewhat complex and discussed in more detail later. Finally, many of the algorithms needed are recursive in nature, such as the evaluation of Wick's theorem, and not all dialects of Fortran allow recursion. The next language to be discussed, Lisp, was actually used in the original implementation of SQSYM. It is available on fewer machines than Fortran, but is still fairly common. Lisp has traditionally been an interpreted language, but compilers are commercially available. Lisp is a good symbol manipulation language, but even when it is compiled it runs more slowly than either Fortran or C. User data types can be constructed as lists of the built-in data types and Lisp will provide for recursion. This is why Lisp was preferred over Fortran. Lisp was chosen as the language for the initial implementation for SQSYM, primarily because of our lack of knowledge about C. However, the Lisp version of SQSYM was too slow and required too much memory. This motivated a rewrite of SQSYM using the C language. This language provides the flexible definition of user data types, recursion, and portability. The C programming language also has compilers available which generate efficient code and has been quite successful in allowing rapid implementation of the numerous requisite algorithms.

At this point we have a compiled program, written in C, which reads a terminal device or disk files to obtain commands which are executed as they are read. Most of these commands are used to describe the data which is to be processed. The first group of commands which must be given are the declarations. The **declare** command must be used to associate a label with a type. For example the two electron integrals, $(ij|kl)$, are an array of scalar numbers. If we wish to have SQSYM associate with them the label "*h*2", then the command "**declare scalar** *h*2" would be given to SQSYM. A list of data types and descriptions of what they mean is given in Table 1.

The next piece of information needed is a description of the indices needed to specify a spin orbital. This is given by the **direct_product** command. The arguments to **direct_product** are the names of the spaces for which the direct product is being taken. Typically, the only spaces needed here are the spin space and the spatial space. Also given with this command is the breakdown of each of the spaces into ranges. The spin space can be decomposed into two ranges each of dimension one, α and β, or one range of dimension two. The spatial orbital basis can be broken down into a particle and hole space. The dimensions of the particle and hole space can be given as **undefined** allowing the code which SQSYM produces to be general. An index for the irreducible representation is not needed for SQSYM. Finite, nondegenerate point group symmetry can be fully taken into account in another way and this will be explained later.

After the breakdown of the spin orbital indices is given, the **range** command is used to specify subspaces within a space. Ranges within a space are given with a binary notation, a zero in the *n*th position indicating that subspace *n*, as defined by the **direct_product** command, is not included in the range and a one in this position tells SQSYM to include this subspace in the range. For example, let "space_*s*" be declared as the space of spatial orbitals. The range specified by "space_*s* 10" could be taken to refer to hole orbitals if the quasivacuum has been appropriately defined in the manner described below. In this case the range "space_*s* 01" would have to indicate particles. The command "**range** holes space_*s* 10" would associate with the label "holes" the subspace "space_*s* 10".

When the expectation value with respect to the quasivacuum is computed, SQSYM must have some idea of what the quasivacuum is. The quasivacuum is specified with the **vacuum** command. The quasivacuum can be given as a direct sum of direct products of ranges. A common choice of vacuum is the direct product of the particles with the full spin, both α and β, space.

Table 1. A list of the data types accepted by SQSYM

Type	Description
scalar	a factor which is not an operator
creation	a creation operator factor
annihilation	an annihilation operator factor
external	an arbitrary index that is not summed over
internal	an index for which an implicit summation applies
space	a basis such as the spatial orbitals or spin functions
range	a range specifies a subspace of a specified space such as particles or holes
expression	the basic mathematical quantity which the user manipulates which is a representation of a sum of terms

Some of the declared labels require that more information be given than just their types. Any label of **scalar** type requires information about how many indices the array has, the permissible ranges of the indices, and the permutation symmetries of the indices. All of this information is required if the resulting expressions are to be simplified to the greatest extent possible. This information is conveyed to SQSYM through the **factor** command. The arguments to **factor** are the label which has already been declared as a factor, a range for each index of the factor, and the permutation symmetries given as products of index transpositions together with the sign change that the permutation causes. For the two electron integrals, $(ij|kl)$, the **factor** command would read "**factor** $h2$ space space space space (+(1 2)) (+(3 4)) (+(1 2)(3 4)) (+(1 3)(2 4)) (+(1 2)(1 3)(2 4)) (+(3 4)(1 3)(2 4)) (+(1 2)(3 4)(1 3)(2 4));". This command would be issued after the label "$h2$" is declared as a factor and the label "space" is declared as a range. Furthermore, the **range** command is required to precisely specify the range of "space" before this **factor** command is issued.

To make it possible to begin constructing expressions, information is needed about the indices, which are used for two purposes. The indices declared as **external** represent an unspecified value and those declared as **internal** always occur in pairs and represent a summation over the range which is associated with that index. The **index** command is used to associate a range with indices.

The last command needed to begin work is **assign**, which performs requested operations on expressions and associates the result with an expression label. The operations are described with a Lisp-like syntax which allows recursive reuse of operations. For example, the **plus** function can be given as one of its arguments the result of another **plus** function call. The format of the assign command is "**assign** expression_label expression". When the interpreter reads an expression it first checks to see if it has read an expression label. If so, an internal table mapping the expression labels to the value of the expression is consulted to obtain the expression. If instead, a right parenthesis is read, then the interpreter takes the next word read to be the name of an internal function. This function is then called and may expect more expressions or other data in its argument list. If more expressions are needed, then the routine which reads expressions is recursively reentered.

The functions with are recognized by the SQSYM **assign** command are summarized in Table 2. The most important of these are **plus**, **times**, **canonicalize**, **substitute**, **combine**, **term**, **vetimes**, **normalorder**, **adjoint**, **delta**, **switch**, and **contract**. The most basic of these is **term**, which allows the user to begin building up expressions by giving terms in a symbolic format. Let "x" and "y" be declared as factors, each of which has two indices, and let "i" be an internal index and "u" and "v" be external indices. Then, the function call "(**term** +1 $x(u\ i)$ $y(i\ v)$)" would produce an expression which is a symbolic representation for the matrix multiply between "x" and "y". The arguments which the **term** function requires is first a constant multiplicative factor, followed by the factors and their indices. The internal indices which have been given in the **term** function must appear twice and they are implicitly summed over the range specified for that index with the **index** command.

Now that the **term** function has allowed us to enter expressions, we can manipulate them with the other functions. The functions **plus**, **times**, **normalorder**, and **adjoint** do exactly as their name implies to their arguments. The other principal data manipulation function is **vetimes**. This function takes several expressions as arguments. The quasivacuum expectation value is computed for

Table 2. The functions recognized by the **assign** command

Function	Description
adjoint	computes the adjoint of an expression
canonicalize	converts each term in an expression to a canonical form
combine	removes duplicate terms
connected	returns the terms with connected diagrams
contract	contracts two external indices in an expression
delta	simplifies delta functions appearing in an expression
linked_disconnected	returns the terms with linked disconnected diagrams
normalorder	converts an expression to its normal ordered form
plus	sums a group of expressions
restore	obtains an expression from a disk file
substitute	replaces factors with expressions in an expression
switch	switches two external indices in an expression
term	accepts a symbolic representation of an expression
times	finds the product of a group of terms
unlinked	returns the terms with unlinked diagrams
vetimes	finds the vacuum expectation of a product of expressions using Wick's theorem
vewick	use Wick's theorem to take a quasivacuum expectation value

the product of these expressions. When a quasivacuum expectation value is to be obtained for a product of expressions, it is not efficient to do the product first and then take the expectation value. The problem is that many of the terms in the product will give zero contribution to the expectation value. Knowing in advance of taking the product that we are going to take the expectation value, a large number of the products need not be formed. For example, in open-shell spin-adapted coupled cluster theory, the eighth power in T is required for some of the equations. Since the T expression has nine terms, computing the product would produce $9^8 = 43{,}046{,}721$ terms. But only two of the nine terms can possibly contribute, meaning that only $2^8 = 256$ terms in the product can contribute to the quasivacuum expectation value. Although it is true that the user of SQSYM could avoid such excessive computation without using the specialized **vetimes** routine, to have to explicitly take advantage of these savings would require long and complex input files, with an increased possibility of error.

The **combine**, **delta**, and **canonicalize** functions do basic simplifications. The first step in the simplification of an expression is canonicalization. This involves taking each term in an expression and rearranging its factors to some unique order. If the factor's indices can be rearranged using the index permutation symmetries that have been given with the **factor** command, then the indices must be rearranged to a unique form as well. Since more than one of a given type of factor can occur in a term, the arrangement of indices within a factor can influence the ordering of factors within a term. Thus, the index rearrangement and term reordering become coupled in a complex way. The **canonicalize** function completely solves this problem; that is, no matter how the factors or indices in a term might be permuted, the **canonicalize** function will put the term in the same canonical form. This is important for the **combine** function, which searches through an expression for terms which are identical. If two identical terms are found, then they are replaced with a single term equal to the sum of the two original terms.

The **delta** function attempts to remove Kronecker delta functions from terms. This can usually be done when one of the delta function's indices is not an external index. When both of the delta function's indices are internal and identical, the delta function can be replaced by the dimension of the range over which the index sums. This is why the dimensions of subspaces can be given with the **direct_product** command. If a dimension was given as **undefined** and a delta function of this sort was encountered, then **delta** command would leave this delta function in the term. Otherwise, the multiplicative factor is adjusted by the appropriate amount.

Some other simplification routines are useful for rearranging the expressions to a form that can be conveniently iterated upon when the time comes to apply the theory to a chemical system. Occasionally, it is desirable to switch two external index labels within an expression. The **switch** command does this, given two external index labels and an expression as arguments. The **substitute** function will search through all of the terms within an expression to find a specified factor, which it will replace with a given expression. Finally, the **contract** function allows two external indices to be set equal to each other and summed over throughout all the terms in the expression.

A summary of the commands used to manipulate and simplify data as well as some other utility commands is given in Table 3.

The manipulation and simplification of expressions are not enough. Several more simple tasks must still be accomplished. These include commands to print

Table 3. The commands accepted by the SQSYM interpreter

Command	Description
>	redirect the output of a command to the named logical file
adjoints	indicates which operators are adjoints of each other
assign	associate a label with an expression computed in a Lisp-like language
autorestore	indicates that all expressions needed by **assign** are automatically restored from disk
autosave	indicates that all expressions computed with **assign** are automatically saved to disk
cvacuum	define the complement to the quasivacuum
declare	associates a label with a type
direct_product	specify the spaces used to index the one particle basis functions
factor	define a factor's indices and permutation symmetries
file	associate a logical file name with a filename
fortran	translates expressions into Fortran 77
index	associate indices with ranges
memory	indicates how much memory SQSYM is using
print	display various quantities internal to SQYSM
range	associates a label with a subspace
relation	establish a relationship between a factor and an expression for the **substitute** function
resource	print out information about SQSYM's resource consumption
vacuum	define the quasivacuum
vadvise	notify the operating system about the paging requirements
write	write a message to the output

Table 4. The commands which interface to the compiler portion of SQSYM

Command	Description
data_address	associates an index into an address array with a factor provided to CORR
dataflow	compile an expression in an intermediate representation and translate this IR into the CORR language
range_address	gives an index into an address array for the ranges stored in CORR
result_address	associates an index into an address array with a factor computed by CORR

expressions, save and recover expressions and other internal data from disk files, and give information about SQSYM's resource consumption. Although these are necessary in such a program their implementation is simple and will not be discussed any further. A less trivial problem is that of converting the final expressions into a computer program which can compute wavefunctions and energies for an arbitrarily chosen molecule. This correlation energy program will be referred to as the CORR interpreter or CORR.

Four commands, which have been summarized in Table 4, have been introduced into SQSYM to compile the expressions into a program. Three of these give SQSYM information about addresses of data that CORR has available or desires to be computed. The address is an integer which is interpreted as an index into an address array. The address array is used to convert the address that SQSYM and CORR have agreed upon to the address where CORR has actually stored the data. The address array is needed because the size of the data depends on the particular case CORR is running; thus the locations of the data can be shifted from calculation to calculation. The address for data that CORR initializes at the start of the run, such as the Hamiltonian matrix elements, is specified with the **data_address** command. Addresses for data which are computed by CORR using the expressions obtained with SQSYM is given with the **result_address** command. Sometimes there is not a direct correspondence between SQSYM's factors and CORR's data. The two electron integrals are represented by a single label, "$h2$", in SQSYM, irrespective of the ranges of the indices. CORR does not necessarily need all of the two electron integrals, however. It may just need the $(ai\,|\,bj)$ block of the integrals, that is the particle, hole, particle, hole block. This is accounted for in the **data_address** and **result_address** commands by allowing the range for the indices to be specified with the address for the data. The other command giving address information is **range_address**. This tells where the actual dimension of a range can be found when CORR is run. This information is necessary for the storage allocation of intermediates. The **range_address** command is also used to tell SQSYM roughly how big a range is. This does fix the dimension of each range, but this information is only used to compute the optimal algorithm for CORR to employ in converting its data into results; the molecules to which the resulting program can be applied are not constrained by this specification.

After all of the needed data and result addresses are given, the compilation of the expressions into an executable program can begin. This is done with the **dataflow** command. The first argument to **dataflow** is a parenthesized list of subcommands and their arguments. The first invocation of **dataflow** requires that the **init** subcommand be specified to prepare SQSYM's internal static variables. The **result** subcommand can be used in the next execution of **dataflow**. The argument to **result** must be a label which was given in a **result_address** command. When the **result** subcommand is given, an expression must be given as an argument to the **dataflow** command. The address of the result is where the evaluated expression is to be accumulated. The external indices in the expression correspond to the indices in the factor specified as the argument to **result**. This subcommand causes the information in the expression to be converted into an internal format for use with the **done** subcommand, which takes this internal format and converts it into a form which can be executed by the CORR interpreter. The information CORR requires is written into two files the names of which are specified using the **datafile** and **modelfile** subcommands. The **datafile** is a text file listing instructions which the CORR interpreter must follow. These instructions are written in the CORR language. The **modelfile** is another text file which contains information about the intermediates CORR will need to use to complete a calculation.

2.3. Data representation

This section delves into a part of the inner workings of SQSYM, specifically, the data representation. The proper choice of the underlying data representation is very important in a program such as SQSYM. It not only provides the framework the program uses to perform the desired work, it also gives the programmer a vehicle to conceptualize the problem. A poor choice would lead to a less comprehensible program and unnecessary difficulty in the debugging process. Since many of the needs of such a symbol manipulator do not become apparent until significant pieces are written, a lack of foresight in the data representation choice could result in rewriting large pieces of the program. The data types chosen will be illustrated with C-like program fragments which contain structure definitions. The structure is one of the user defined data types provided by C and is all that is needed here. The syntax of the structure definition is fairly straightforward; it is possible to get a basic understanding of the data types without understanding much about the C programming language. If more information about C is needed, Kernighan and Ritchie [38] provide a short yet complete description.

The most basic data type describes a range. This is the **range_t** type and is illustrated in Fig. 1. It is built of two structure members. The first is an integer which specifies the space the range applies to, usually either space or spin. The actual value of this integer is a number internally assigned by SQSYM to the space label at the time the label was declared. In fact, all labels are assigned integers at declaration time. These integers provide a compact way to refer to the

```
typedef
struct {
  int space; /* The space which this range occupies. */
  int sub; /* The subspaces within space which this range covers. */
  } range_t;
```

Fig. 1. The **range_t** type definition. An index may take on values represented by this structure

labels during the manipulations performed by SQSYM. The second structure member in the **range_t** type is an integer which gives the subspaces included in this range. The subspaces are given as a string of bits. A one indicates that the subspace corresponding to that bit position is included and a zero means that that subspace is not included.

The **factor_t** type, depicted in Fig. 2, is designed to describe a factor as it would appear in a term. The integer members **n**, **t**, and **id** give the number of indices, the factor type, and the factor identifier, respectively. The factor type is either scalar, creation operator, or annihilation operator in the current implementation. Such things as unitary group generators could be added in future versions of SQSYM. The factor identifier distinguishes among the different factors of a given type. For example, the Hamiltonian and cluster operator are both scalar factors, so their **t** members are identical, but their **id** fields are different and used to distinguish them. In addition to these members describing the factor, three arrays are needed to describe the indices. The first array, **f**, is of integer type and gives a pointer to another factor within the term. This index is contracted with an index within the factor pointed to by the **f** array. If the pointer is invalid, then the index is understood to be external, rather than internal. The second array, **i**, is also of the integer type. It contains a pointer to the location of the index within the factor pointed to by **f** with which the current index is contracted. If **f** indicates that this factor is external, then **i** contains the integer corresponding to the label of the external index which belongs here. Lastly, an array of the **range_t** type is needed to specify the ranges of the indices. This factor type can now be used to construct a term.

A term consists of a product of factors with summations over pairs of indices. The data type which represents terms is **term_t** and has been shown in Fig. 3. The structure members that are needed to describe a term are **n**, the number of factors; **num**, an integer giving the numerator of the constant multiplicative factor; **den**, the integer denominator; and an array of length MAX_FACTOR of structures of the **factor_t** type. The ordering of factors is important since some of the factors can be operators.

Finally, expressions can be represented by grouping terms together. Since the number of terms in an expression varies widely, a linked list is much more desirable than an array. However, it was feared that increased randomness in memory access and extra overhead in memory allocation and deallocation would impair the performance of SQSYM if a simple linked list were used, so expressions were implemented as a linked list of small arrays of terms. This data

```
typedef
struct {
  int n; /* The number of indices. */
  int t; /* The factor type of the factor. */
  int id; /* The index of the factor in the declaration list. */
  int i[MAX_INDICES]; /* Pointer to contracted index. */
  inf f[MAX_INDICES]; /* Pointer to the contracted factor. */
  range_t r[MAX_INDICES]; /* The ranges of the indices. */
  } factor_t;
```

Fig. 2. The **factor_t** type definition. This type of data describes a factor as it would appear in a term. The symbol MAX_INDICES is a constant specified at compilation time and limits the number of indices which may appear in a factor

```
typedef
struct {
  int n; /* The number of factors in this term. */
  int num; /* The numerator of the constant multiplicative factor. */
  int den; /* The denominator of the constant multiplicative factor. */
  factor_t f[MAX_FACTORS];
  } term_t;
```

Fig. 3. The **term_t** type definition. This describes a term according to its constituent factors. The MAX_FACTOR symbol is replaced by a constant at compile time. The value of this constant limits the number of factors which can appear in a term

```
struct expression_struct {
  int n; /* The number of terms. */
  int protect; /* Do not purge this expression when done if set to not 0. */
  struct expression_struct *p; /* A pointer to the next block of terms. */
  term_t t[N_TERM];
  };

typedef struct expression_struct expression_t;
```

Fig. 4. The **expression_t** type definition. This is a linked list representation of a sum of terms and is the basic quantity which a user of SQSYM manipulates

type is called the **expression_t** and is depicted in Fig. 4. The members of this structure are an array of terms, **t**, of length N_TERM; the number of terms, **n**, in the array **t**; and the pointer to the next **expression_t** in the linked list, **p**. When manipulating expressions it is important to deallocate memory for all expressions which are no longer to be used. Expressions which do not correspond to labels may be deallocated as soon as they are used. However, SQSYM's **assign** functions do not know where its expression arguments came from. Thus, it is necessary to have an additional member in the expression, the **protect** member. This is one if the expression is to be kept after its use and zero otherwise. When an expression is assigned to a label, all the the **protect** members are set to one, until another expression is assigned to that label, at which time the old expression is deallocated without regard for the **protect** member. This is the only time that the **protect** member is ignored.

Although this is enough to describe the most important aspects of SQSYM, we have found this scheme to be very memory intensive. For example, when combining terms, all of the terms within an expression must be rapidly examined. If the entire expression cannot be held in the central memory of the computer, then much time is spent accessing the peripheral paging devices. To avoid this problem the **indexed_expression_t** type has been introduced. This type and the associated **expression_index_t** type are shown in Fig. 5. The **indexed_expression_t** is implemented like **expression_t** with a linked list of arrays. In this case the length of each array is MAX_INDEXED_EXPRESSION. To place a term in an **indexed_expression_t** we start by assigning every term an index or hash. The procedure used to obtain the index is free to be chosen, but it should break the terms apart into several different classes of terms and it must be possible to

```
typedef
struct {
  int n; /* The number of factors. */
  int t[MAX_FACTORS]; /* The factor type of the factor. */
  ind id[MAX_FACTORS]; /* The index of the factor. */
  } expression_index_t;

struct indexed_expression_struct {
  int n; /* The number of expression groups in this block. */
  expression_index_t i[MAX_INDEXED_EXPRESSION];
  expression_t *e[MAX_INDEXED_EXPRESSION];
  struct indexed_expression_struct *p; /* Pointer to next block. */
  };

typedef struct indexed_expression_struct indexed_expression_t;
```

Fig. 5. The **expression_index_t** and the **indexed_expression_t** types. The former data type is a hash for terms within an expression. The latter stores the hash chains for the needed hashes

quickly determine the index of a term and compare this to the indices of other terms. The expression indices in SQSYM currently use the order and types of factors in the term to generate an index. When a term is added to an indexed expression, the list of expression indices is scanned until a match with the index for the term is found. If a match is found, the term need only be compared with the terms in the expression associated with the matching expression index to sum it into the entire expression. Otherwise, a new expression index is added to the indexed expression list and the term is associated with this new index. In this scheme, only a small subset of the terms must be examined to find out if the new term matches a term in the expression and, consequently, this method greatly relieves the strain on the computer's memory.

Many other data types are used in the internal working of SQSYM, but all of these details are not particularly interesting. Our principal goal is to provide enough information to allow a C programmer to produce a symbol manipulator oriented towards the second quantization formalism without having to do any major backtracking or trailblazing throughout the development process. To this end some hindsight should be added here to warn programmers about the disadvantages of SQSYM. Of course, it is easy to think of equations to derive which exceed the limitations of available machines. The symbol manipulation algorithms in SQSYM are fairly efficient when the required processing time is considered; however, we have found the required memory to be very large, even on a MIPS 2000/8 computer equipped with 64 megabytes of central storage. This does not simply refer to the total memory the program uses, but rather the amount of central memory needed to prevent paging from becoming a major bottleneck. Some runs of SQSYM could require a total of 150 megabytes, but do not cause a very heavy paging load on the system, while other runs could use 80 megabytes total but access all of these data in very rapid cycles, causing heavy demands on the paging subsystem. One simple way to reduce SQSYM's memory requirement is to use 8 or 16 bit integers where possible instead of the standard 32 bit integers and this has been implemented. A more difficult approach would be to avoid fixing the dimensions of the arrays, but, instead, replace the arrays with pointers to the data and make sure that just enough memory is allocated to hold the data. The latter approach would increase the overhead associated with accessing, allocating, and deallocating the data and for these reasons was not chosen for use by SQSYM. Going back to modify SQSYM now would involve considerable work, but for a programmer starting from the beginning, more memory-efficient approaches are worth consideration, at least.

SQSYM is a fairly large and complex program and simplicity of approach should always be strived for. Unfortunately, simplicity sometimes conflicts with efficiency. For example, in the index and factor pointers in **factor_t** type there is some duplication within all of the factors in a **term_t** data type. To eliminate this redundancy within a **term_t** would probably sacrifice much of the simplicity of the algorithms and should be avoided if possible.

2.4. Data manipulation

As with the previous section on data representation, this section will discuss matters internal to the SQSYM program, specifically, the principal algorithms used to manipulate the data. Many of the tasks performed by SQSYM are quite simple. For example, the addition of two expressions requires only that the linked lists corresponding to the two expressions be joined together. Since we chose a linked list of arrays of terms as the representation for an expression, it is also desirable to copy a few terms into the empty slots at the end of one of the expressions to allow deallocation of the last **expression_t** link in the other expression if all of the slots in this link become empty after the copy. The multiplication of two expressions is also very simple, due to the representation chosen for terms. The product of two terms involves only copying the factors from the second term to positions after the factors in the first term and then offsetting the factor pointers, **f**, for the second term by a constant amount. Then the constant multiplicative factors are multiplied and simplified to remove common factors from the numerator and denominator. We see how the careful choice of data representation can make basic tasks like these routine. However, no choice of data representation can make everything simple. We still need to invest some effort to do certain tasks, the most difficult of which are taking the quasivacuum expectation value and term canonicalization.

The quasivacuum expectation value is taken with the assistance of Wick's theorem. Wick's theorem provides a simple way to rewrite a product of creation and annihilation operators as a sum of normal ordered products of operators multiplied by Kronecker delta functions. Since the quasivacuum expectation value of a normal ordered operator is zero, only the terms which contain no operators survive. As mentioned earlier, when the expectation value of a product of expressions is needed, it is best to form the product between expressions at the same time as the quasivacuum expectation is taken.

The procedure begins by setting up arrays describing the operators in each term of the expressions for which we need the product. This description is expressed with the type represented in Fig. 6 and is known as the **excitation_count_t** data type. Its members include **vmin** and **vmax** which respectively store the minimum and maximum number of excitations this term can produce relative to the quasivacuum. Also included are the minimum and maximum number of excitations relative to each subspace, **smin** and **smax**, respectively. The reason the minimum and maximum excitation counts are not equal is that the indices of the creation and annihilation operators can range over several subspaces. For the

```
typedef
struct {
  int vmin; /* min number of excitations relative to the vacuum */
  int vmax; /* max number of excitations relative to the vacuum */
  int smin[MAX_SPACE][MAX_SUBSPACE]; /* relative to subspace */
  int smax[MAX_SPACE][MAX_SUBSPACE]; /* relative to subspace */
  } excitation_count_t;
```

Fig. 6. The **excitation_count_t** type. This stores information about the action of operators within terms and expressions

expressions that are being multiplied an overall excitation count is determined for each by examining the excitation count for all of the terms within each expression. Finally, starting from the leftmost expression in the product, the cumulative excitation count is obtained for each expression.

With this information on hand we can begin efficiently building the product of the expressions. The first term from the rightmost expression is chosen and its excitation count is compared to the cumulative excitation count. If the cumulative excitation count for the expressions from the left can potentially negate the cumulative excitation count for the terms from the right, then the next term in the product is selected from the next expression to the left. Otherwise, the contribution to the product involving all of the terms selected so far will be zero, since by the time we get to the quasivacuum on the right we must have something which will annihilate it. In this case another term is selected until one which gives a nonzero contribution is found. If such a term is found, the process proceeds to the next expression to the left of the expression containing this term.

At some point a term from each expression may be selected such that the quasivacuum expectation of their product will be nonzero. This term is handed to a routine which computes the fully contracted part of Wick's theorem and, hence, obtains the quasivacuum expectation value. The routine rewrites the term in such a way that all of the fully contracted terms produced by this term are efficiently found. For the purposes of finding the contractions the operators are the only factors that are needed. Furthermore, whether an operator is a creation or annihilation operator is not as important as its action on the quasivacuum. Operators are classified as $\boldsymbol{R}$ or $\boldsymbol{L}$, indicating whether they annihilate the quasivacuum from the left, $\boldsymbol{R}|0\rangle = 0$, or the right, $\langle 0|\boldsymbol{L} = 0$. Some operators can annihilate the vacuum from both directions, because the range of their indices can span more than one subspace, so these can be $\boldsymbol{R}$ or $\boldsymbol{L}$ operators. Arrays are formed giving the action of each operator upon the vacuum and the type of each operator, which is either creation or annihilation in the current implementation of SQSYM, and this is all the data that is needed to begin forming the contractions. Wick's theorem states that contractions are formed only between pairs of operators with an $\boldsymbol{R}$-type operator to the right and an $\boldsymbol{L}$-type operator to the left. The contraction routine starts by looping through all available pairs of $\boldsymbol{L}$ and $\boldsymbol{R}$ operators. For each pair it recursively calls itself to find the next pair. When it finds that no operators are left, it calls another routine which converts a list of contractions into a term and adjusts its sign appropriately. The term is then canonicalized and summed into an indexed expression. Canonicalization and summing must be done as soon as possible to prevent an enormous scratch expression from being formed which would have many redundant terms.

The canonicalization procedure implemented in SQSYM is exhaustive. That is, if two equal terms which are not written in the same form are presented to it, then they are guaranteed to be rewritten in identical forms. Here, as in the Wick's theorem routine, a representation for the term must be chosen which is more suitable for this procedure. It is convenient to stop viewing the term as a product of factors which have indices contracted with indices in other factors and start viewing the term as a single factor, the superfactor, with many indices. To form the superfactor the scalar factors in the term must first be reordered to a standard form. This is just a simple sort; the relative ordering of factors which have identical **ids** is not yet a concern. The superfactor is then formed. Three arrays are used to describe the superfactor. The index array is equivalent to the **i** member of the **factor_t** data type. The range array is equivalent to the **r** member of the **factor_t** type. There is no equivalent to the **f** member of **factor_t**, because the term consists of only one factor in the superfactor representation. Finally, we need the index permutation symmetries of the superfactor. This permutation group is not simply the direct product of the permutation groups of the constituent factors, because there is a possibility that a given type of factor appears twice in the term, so we must include the direct product of the symmetric groups describing the permutation of sets of indices among the identical factors. Now we are ready to search for the canonical form of the term.

The search begins by looping through all superfactor permutations. Each permutation is applied to the superfactor index array. The permuted index array is then compared to the most canonical index array found so far. The most canonical index array is initially set to the unpermuted superfactor index array. If a permuted index array is more canonical than the most canonical index array, then the most canonical index array is replaced by the permuted index array. After all permutations are exhausted the most canonical index array contains the canonical superfactor. The definition of "more canonical" is somewhat arbitrary. The method used by SQSYM is to compare the permuted index array to the most canonical index array, index by index, starting from the left. If the value of the index array member for the permuted index is less than the value of the most canonical index, then the permuted superfactor is more canonical than the most canonical superfactor. If the values are the same, then the next indices to the right are considered. If all of the indices are identical, then the range arrays are examined in a similar manner. Since each index array member is repesented as an integer and each range array member is expressed as two integers, the comparison of these members is a computationally efficient process. For canonicalization to be truly exhaustive there is one restriction on the term. All of the ranges in the term must be a simple subspace. For example, a range may be particle or hole but cannot indicate a summation over both subspaces. This restriction is not intrinsic to the canonicalization algorithm; it is only needed if all equivalent terms are guaranteed to be transformed into identical forms.

The number of superfactor permutations can become quite large, with several thousand permutations commonly needed. Less computationally demanding algorithms could be developed which do not rigorously canonicalize the term. The faster algorithm could be used to combine and eliminate the majority of terms and then a complete canonicalization could be performed to finish the simplification of the expressions. However, it has been found that canonicalization is not the limiting step in SQSYM. The time spent in various routines during typical calculations was obtained and showed that canonicalization was second to the quasivacuum expectation value in processor time used during the symbol manipulation process. Translation of the equations into an executable form was found to require more processor time than the canonicalization and quasi-vacuum expectation processes combined.

2.5. Executable code generation

The simplified expressions generated by SQSYM can contain thousands of terms. It would be difficult to write a program which uses these equations to compute the properties of a molecule. Thus, SQSYM must further process the expressions and compile them into a form that is suitable for execution. One possible approach would be to convert the expressions directly into Fortran. An advantage of Fortran is its wide availability, including vectorizing and parallelizing dialects. Compilation into Fortran was, in fact, first implemented and still remains an option; however, difficulties were encountered. The Fortran programs produced were so long that compilation times for the Fortran compiler were significant and in some cases the compiler even abended due to the length of the code. Currently, Fortran is avoided as the target language of SQSYM. Rather, a custom language has been developed which simply and compactly describes the computation of an expression. This language will be called the correlation energy language, CORR, and the CORR interpreter is the program which will execute the file containing the CORR language and compute the correlation energy.

The commands which must be a part of the CORR language are best understood by first examining the procedure for converting an expression into an executable program. Let us start by considering an expression with one term:

$$a_{stuv} = x_{uijk} y_{vijl} z_{klst}. \tag{2.14}$$

Assuming that each index ranges from 1 to n, the result, a, can be most straightforwardly computed with $2n^8$ floating point multiplications. This would be done by using a loop for each index. However, a considerably more efficient approach exists. Suppose we break the formation of a into two binary products. The first binary product forms an intermediate array which is then contracted with the remaining array to form a. Table 5 illustrates the possibilities. This table lists the binary product being considered, the result of forming the binary contraction, the number of floating point multiplications required, the amount of memory needed for intermediates, and the cumulative number of multiplications required for the algorithm. The least expensive routes use a total of $2n^6$ floating point operations and require intermediate storage for n^4 floating point numbers. One of these least expensive routes involves first forming the intermediate α by

Table 5. The computational complexity for various evaluation routes for a sample term

Binary product	Result	Mult.	Memory	Cum. mult.
$x_{uijk} y_{vijl}$	α_{ukvl}	n^6	n^4	n^6
$x_{uijk} z_{klst}$	β_{uijlst}	n^7	n^6	n^7
$y_{vijl} z_{klst}$	γ_{vijkst}	n^7	n^6	n^7
$\alpha_{ukvl} z_{klst}$	a_{stuv}	n^6	–	$2n^6$
$\beta_{uijlst} y_{vijl}$	a_{stuv}	n^7	–	$2n^7$
$\gamma_{vijkst} x_{uijk}$	a_{stuv}	n^7	–	$2n^7$

contracting x with y and then forming the result a by contracting α with z. Given a term, SQSYM's **dataflow** command will examine all of the possibilities and choose the least computationally intensive method. If two terms have the same computational complexity, then the method requiring the least space for the intermediates is chosen. The principal deficiency of the term optimization algorithm implemented in SQSYM is that index permutation symmetries are ignored. However, no implementations of such an algorithm will be able to guarantee that the generated procedure will have minimal computational complexity. This is because exact information about the ranges of the loops are not known at compile time. Furthermore, the use of point group symmetry considerably complicates the matter. A good job of optimizing the terms can be done without all of this information by giving SQSYM an approximate dimension for the ranges of the loops. These dimensions are represented by two integers, i and j, and are taken to be in^j. The integer n is unspecified and its precise meaning is arbitrary. We typically take n to be the number of occupied orbitals, so i and j both equal one for the hole orbitals. Since the number of particle orbitals is usually larger than the number of hole orbitals for a reasonable basis set, i is set to two for particles while j remains one. In open-shell triplet calculations the size of the range for the singly occupied orbitals would be two; thus i would be two and j would be zero.

Unfortunately, many more considerations arise when we need the optimal algorithm for computing the result of an expression consisting of more than one term. This is because some of the intermediates may have already been computed and if they are retained in storage instead of deallocated after their first use, they can be used again at the expense of an increased memory requirement. The optimal algorithm for computing a single term may no longer be optimal when the entire expression is considered. Another algorithm may involve an intermediate which has been computed beforehand and might become less computationally expensive when the entire expression is considered in the optimization process. Thus, the optimization of a term in an expression becomes coupled to the optimization of all other terms in that expression. In fact, the problem is even worse than this; the optimal algorithm depends upon all terms in all of the expressions which are to be evaluated, not just the expression in which the term under study appears. The complete minimization of the computational complexity for the evaluation of a set of expressions has not been implemented in SQSYM. Even if an exhaustive minimization were implemented, the processor time required would prohibit its use in all but the simplest cases.

The simplified approach taken in SQSYM is invoked with the **dataflow** command. This command is first used with an initialization option to prepare the dataflow routines. Following initialization, **dataflow** can be executed any number of times with expressions as arguments. These invocations build up an intermediate language which contains information about the expression. The intermediate language is implemented with data structures consisting of the members, **op**, **arg1**, **arg2**, and **result**. The members **arg1** and **arg2** specify the addresses of the data which are to be used and **result** gives the address of the data to be created. In general, the ordering of **arg1** and **arg2** is important. The **op** member is assigned to one of four operations to be performed upon the arguments, **contract**, **accum_delta**, **free**, and **ptrace**. Sometimes these operations require supplemental information which is stored in the **detail** member.

The **contract** operation indicates that a binary contraction between the two arrays is to be done. The **detail** member is used to specify the external indices and the indices to be contracted. The **contract** operation generates a new piece of data, the address of which is placed in **result**.

When the intermediate language has used this temporary datum for the last time it can be deallocated with the **free** operation which takes one argument and produces no result. The actual deallocation of the data does not take place in SQSYM, since the data is never allocated in SQSYM. The storage is really allocated and deallocated when the CORR interpreter is run; however, the **dataflow** component of SQSYM must keep track of the data that the CORR program is going to manipulate, so that memory can be efficiently used when CORR is run.

The **accum_delta** operation takes one argument and sums it into the result. This is very different from the **contract** operation which creates a new result. The result of **accum_delta** is preallocated and is specified as an argument to the **dataflow** command. The **accum_delta** operation is so named because it also allows the accumulation of data which can be written as an array or the product of an array and a delta function between external indices. Products of delta functions between external indices and arrays sometimes arise and handling these cases directly with **accum_delta** avoids the waste of central storage which would result from actually forming the product of the delta function and the array before the accumulation step.

The **ptrace** operation takes one argument and allocates a result. It forms contractions over pairs of indices within the single argument. Given in the **detail** member are the lists of external and internal indices along with which pairs of internal indices are contracted.

When the dataflow routines optimize a term, all possible ways of forming the result through binary contractions of factors and intermediates are considered. As each contraction is considered the dataflow structure is inspected to see if that contraction already exists. If so, the previously computed intermediate can be used instead of the **contract** operation and the processor time that would have otherwise been needed for this contraction is not added to the computational complexity for this term. This type of optimization is known as common subexpression elimination.

The dataflow structure is organized into levels. Level zero consists of statements in the intermediate language which define the known arrays, such as the Hamiltonian matrix elements and the cluster coefficients from a previous iteration. Level one contains statements which depend on data in level zero; level two's statements depend on data produced in level one and the data defined in level zero; and so on, but no level uses data which is generated on the same or following levels. The rationale for this sort of organization is to allow parallel execution of the statements on the same level. Whether or not parallelism is achieved depends on the implementation of the CORR interpreter. The current CORR interpreter is written in Fortran 77 and with some modifications could be parallelized on any shared memory multiprocessor running the UNIX operating system or on multiprocessor IBM S/370 machines if a parallelized dialect of Fortran [39] is installed. Current implementations of the CORR interpreter do not use parallelization. In this case, it is best to generate only one CORR statement per level, since this will allow deallocation of scratch arrays at the earliest possible time and will lighten the memory requirements.

After all terms have been represented in the dataflow structure, the **dataflow** command is invoked with options which tell it to convert the intermediate representation into the CORR language and place it in an output file. The

CORR language is nearly identical to the intermediate representation, except it is converted into a text representation so that the same CORR language files can be ported to a variety of computers. The **contract**, **free**, **accum_delta**, and **ptrace** operations respectively translate directly into the CNTR, FREE, ACDL, and PTRA commands in the CORR language. However, an additional command must be added if parallelized implementations of the CORR interpreter are desired. This command is LEVL, which can be used by the interpreter to determine which statements can be executed simultaneously. The **dataflow** command produces the LEVL command whenever it begins translating a new level in the dataflow structure.

Although the CORR language program produced by SQSYM is optimized to some extent, there are many things which can be done to improve the execution time of the CORR program without modifying the CORR interpreter. The only optimization of the dataflow structure currently implemented is full term optimization with common subexpression elimination. The contractions between factors and intermediates have commutative and associative properties which can be used to further simplify the dataflow structure. These properties are not yet utilized in SQSYM. Also, even though the number of statements on a given level is kept at one to minimize waste of memory, the memory is not used as efficiently as possible. This is because rearrangement of the dataflow levels can result in a smaller amount of memory dedicated to intermediates at any given time. The optimal rearrangement of levels would require the least amount of memory. Rearrangement to the optimal ordering of levels would be extremely difficult. One reason for this difficulty is that actual dimensions of each of the subspaces are not known at translation time. Another problem is that the way the CORR interpreter deals with memory may create areas of temporarily unusable memory in the process of repeatedly allocating and deallocating memory, thereby increasing the total memory requirement in ways unknown to SQSYM.

Finally, even if approximations are made about the dimensions of the arrays and the memory allocation and deallocation procedures that CORR uses, the number of levels can become enormous. For example, suppose we are optimizing the CORR language program for execution on a uniprocessor machine. Then, it is desirable for each level to contain only one operation, since parallel execution is not of use on a uniprocessor machine and the use of only one operation per level allows the most flexible optimization of memory use. However, some of the dataflow structures contain very many operations. For one of the open-shell coupled cluster techniques discussed in Sect. 3 there are over 15,000 operations, making even approximate optimization a formidable computational task. Optimization of the use of memory is not performed in the current implementation of SQSYM, except that intermediates are deallocated after their last use. No attempt is made at level rearrangement. When more powerful machines are at our disposal, once again work along these lines will continue.

2.6. The CORR *program*

We have learned how to generate a partially optimized CORR language program. Now a program, called the CORR interpreter, must be developed to execute this language. Another language must now be chosen to implement the CORR interpreter and we chose Fortran 77 because it is available on all of the machines to which we have access. In addition, our research group has already developed Fortran language programs to compute and write to disk the matrix elements of the Hamiltonian, although it should be pointed out that the use of Fortran has greatly obfuscated many of the algorithms employed. Other languages, such as C and object oriented versions of C, should be considered by anyone beginning a project like this from scratch.

The CORR program is actually a fairly short and simple program. Most of the difficult work has been moved out of CORR and into a library, DTLIB, which provides a variety of routines for manipulating multidimensional arrays of real data, such as the Hamiltonian and cluster coefficients. The DTLIB routines allow the arrays to have index permutation symmetries and point group symmetry, with the restriction to finite nondegenerate point groups. Cole and Purvis [40] have developed a routine similar to one of the primary DTLIB routines, namely that routine which takes the contraction of two arrays. Their implementation does not permit index permutation or point group symmetries; however, it enjoys the advantage of being able to work with arrays that are stored on disk devices. The DTLIB library requires that all arrays be kept in the fast central storage of the computer, although, with some more work, these arrays can be disk resident and efficiently accessed. This work is planned for future revisions of the DTLIB library.

Before CORR is executed, another program, SORTER, must be run to sort the Hamiltonian into the format that DTLIB uses. When CORR starts up, it reads in the arrays produced by SORTER. All the addresses of these arrays are kept in another array, the address array. The commands which CORR interprets refer to locations in the address array, rather than the address of the array itself, since these addresses vary from case to case. Correspondences between locations in the address array and the factors are made known to SQSYM through the **data_address** and **result_address** commands which interface with the compiler portion of the SQSYM interpreter/compiler. The CORR program must make identical correspondences when it initializes the address array. After CORR completes its initialization it begins interpreting the CORR language file produced by SQSYM. After the interpretation is finished, the result of the expression evaluation is placed in the array pointed to by the address array elements specified in the **result_address** command. For all of the correlation energy calculations which CORR currently can do this represents only one iteration. The result of evaluating the expressions is used to update the wavefunction guess for the next iteration. For example, in coupled cluster theory, the resulting expressions should be zero when the equations are converged. If the root mean square of the result coefficients is above some tolerance, then another iteration begins. The cluster coefficients for the next iteration are obtained by absorbing the error in each result into the cluster coefficient which is dominant for that result. Note that the CORR language has no provisions for iteration or updating the cluster coefficients. These could be included, but the CORR language is probably best kept as simple as possible. Since extrapolation methods will be eventually incorporated into CORR, it would be desirable to be able to take advantage of these methods without rewriting the CORR language file, which is most easily done if the CORR language does not explicitly iterate.

It has been mentioned earlier that execution of the CORR program could be accelerated if SQSYM would more thoroughly optimize its intermediate language and, thus, the CORR language program into which it is translated. However, the performance of CORR is not solely determined by the quality of code that SQSYM produces. The CORR program does time-consuming operations on large pieces of data and the performance of the CORR interpreter for a given CORR language input must also be scrutinized. The SQSYM program and the CORR program both allow index permutation symmetries to be specified, but, due to limitations in both SQSYM and CORR, the intermediate arrays produced by the CNTR command, which forms a contraction, have no index permutation symmetries. This wastes both processor time and storage. A benefit provided by CORR is the efficient method it uses to form contractions. First, the arrays involved are repacked in a way that allows the contraction to be expressed as a matrix multiply. After the matrix multiply, the inverse of the packing procedure is applied to place the resulting matrix into a multidimensional array. This is done one symmetry block at a time, so full advantage of point group symmetry is taken. This method does introduce overhead associated with repacking the arrays, but, for typical electronic structure theories, the ratio of overhead to processor time would tend to zero as the size of the system studied tended towards infinity. For example, in CCSD, the largest array has a size that goes roughly like n^4, where n is the number of doubly occupied orbitals. Thus, the overhead associated with repacking the array increases with n in a manner no worse than n^4. However, the most computationally intense terms increase like n^6; thus, for large enough n, the overhead associated with repacking becomes insignificant. Also, the most computationally intense term becomes a matrix multiply between two matrices of dimension n^2. A matrix of this size is large enough to make efficient use of vector architectures as well as allowing the use of fast matrix multiply algorithms [41] which can reduce the n dependence of the processor time needed for these terms from n^6 to $n^{5.61}$ or less.

3. Applications of SQSYM

The symbol manipulator discussed in Sect. 2 has been applied to a variety of electronic structure theories, some well-known and some not yet discussed in the literature. The better-known methods include configuration interaction with single and double (CISD) excitations from a single reference. For this case SQSYM was used to produce spin-adapted equations for closed-shell and high-spin open-shell (HSOS) types of reference functions. These methods are presented to further illustrate the use of SQSYM. The correlation energies produced by SQSYM/CORR for these methods agree with energies generated independently with a graphical unitary group approach (GUGA) CI program [42]. The spin-adapted closed-shell coupled cluster case has also been investigated and SQSYM/CORR was found to produce correct results for this case as well; however, this will not be presented in the present work.

The extension of CCSD theory to the HSOS case is of primary interest in this work and three avenues have been pursued along this line. The first is the implementation of the method of Rittby and Bartlett [28] where a spin unrestricted cluster operator acts upon a restricted Hartree–Fock (RHF) reference function which is of the HSOS type. Spin contaminated contributions to the energy do not occur with this technique, but the equations for the wavefunction are spin contaminated. Second, a method which uses a cluster operator which is chosen using the criteria set forth by Nakatsuji and Hirao [29]. While this cluster operator does not produce spin contamination when operating upon the reference wavefunction the cluster operator does produce states of incorrect spin symmetry when its square operates upon the reference. The spin contamination is completely projected out in the equations for the wavefunction and energy in this case. Finally, a method which uses a cluster operator which commutes with S^2 is presented. No spin contamination arises at any point in this method.

3.1. The closed-shell CISD *equations*

Let $|0\rangle$ be a closed-shell RHF reference function. The CI wavefunction, $|\Psi\rangle$, can be written as a linear combination of $|0\rangle$ and excitations thereof, $X|0\rangle$:

$$|\Psi\rangle = |0\rangle + X|0\rangle, \tag{3.1}$$

where $|\Psi\rangle$ is normalized such that $\langle\Psi|0\rangle = 1$. The spin-adapted excitation operators for CISD can be written in terms of the unitary group generators,

$$E^p_q = \sum_\sigma^{\alpha\beta} a^\dagger_{p\sigma} a_{q\sigma}, \tag{3.2}$$

to form a linearly independent, although nonorthonormal, set of excitations from the reference:

$$X = X_1 + X_2, \tag{3.3}$$

where

$$X_1 = c^a_i E^a_i, \tag{3.4a}$$

$$X_2 = c^{ba}_{ji} E^b_j E^a_i, \tag{3.4b}$$

and the c^a_i and c^{ba}_{ji} are arrays which are yet to be determined. The indices i and j refer to orbitals which are doubly occupied in the reference and the indices a and b refer to those orbitals which are not occupied in the reference. The c arrays possess the index permutation symmetry

$$c^{ba}_{ji} = c^{ab}_{ij}. \tag{3.5}$$

The Schrödinger equation, in terms of the normal-ordered Hamiltonian, may be written

$$H_n|\Psi\rangle = E_{\text{corr}}|\Psi\rangle. \tag{3.6}$$

The energy can be computed by projecting both sides of this equation by $\langle 0|$ to get

$$E_{\text{corr}} = \langle 0|H_n|\Psi\rangle. \tag{3.7}$$

The equations for the c arrays are determined by projecting Eq. (3.6) from the left by the singly and doubly excited states:

$$E_{\text{corr}}\langle 0|E^i_a|\Psi\rangle = \langle 0|E^i_a H|\Psi\rangle \tag{3.8a}$$

and

$$E_{\text{corr}}\langle 0|E^j_b E^i_a|\Psi\rangle = \langle 0|E^j_b E^i_a H|\Psi\rangle. \tag{3.8b}$$

However, these equations are not yet in the correct form for iterating upon. For a given choice of i, a, b, and j the projection for X_2 contains large contributions from two c elements, c^{ba}_{ji} and c^{ba}_{ij}, both of which multiply diagonal elements of the Fock matrix. There are two ways to get a large contribution from only a single c element; we can take linear combinations of the above equations or we can change the projection scheme. Changing the projection scheme has the side effect of simplifying the derivation of the equations as well. A possible choice for the new states which project out the c equations are the determinants

$$a_{a\alpha}^{\dagger} a_{i\alpha} |0\rangle \tag{3.9a}$$

and

$$a_{a\alpha}^{\dagger} a_{i\alpha} a_{b\beta}^{\dagger} a_{j\beta} |0\rangle \tag{3.9b}$$

which are not eigenstates of electron spin. Equation (3.9a) can be expressed as a linear combination of

$$E_i^a |0\rangle \tag{3.10a}$$

and

$$(a_{a\alpha}^{\dagger} a_{i\alpha} - a_{a\beta}^{\dagger} a_{i\beta}) |0\rangle. \tag{3.10b}$$

The latter of these states is a triplet. Thus, when it projects upon any singlet spin function, such as $|\Psi\rangle$ or $H|\Psi\rangle$, it will give a contribution of zero and we are free to take linear combinations of this triplet state with the original symmetry adapted state to obtain a state which can be used to project the equations. In a similar manner, the above doubly excited determinants, Eq. (3.9b), can be obtained by linear combinations of states with states of spin symmetry different than the reference as well as states of the correct spin symmetry. When linear combinations of states of the correct spin symmetry are used to form a projector, it is important to only transform among the set of states which were in the original set of projectors. This restriction is trivial to observe in the closed-shell case; the only way to violate it is to include triply excited states. However, in the open-shell case the excitations are limited to the interacting space and it is easy to accidentally project upon an undesired state, even though it is of the appropriate spin symmetry and appears to have only double excitations in it. This will be discussed in more detail below.

3.2. The high-spin open-shell CISD *equations*

The problem to be solved here is exactly the same as in the closed-shell CISD case, except the reference function now contains some orbitals with only α spin electrons. The excitations which interact with the reference through the Hamiltonian are

$$E_i^a |0\rangle, \tag{3.11a}$$

$$E_x^a |0\rangle, \tag{3.11b}$$

$$E_i^x |0\rangle, \tag{3.11c}$$

$$E_j^b E_i^a |0\rangle, \tag{3.11d}$$

$$E_x^b E_i^a |0\rangle, \tag{3.11e}$$

$$E_j^x E_i^a |0\rangle, \tag{3.11f}$$

$$E_j^y E_i^x |0\rangle, \tag{3.11g}$$

$$E_y^b E_x^a |0\rangle, \tag{3.11h}$$

and

$$E^y_j E^a_x |0\rangle, \tag{3.11i}$$

where the x and y indices refer to those orbitals which are singly occupied in the reference. To form the CISD wavefunction coefficients must be associated with these exictation operators. These coefficients are

$$c^a_i, \tag{3.12a}$$

$$c^a_x, \tag{3.12b}$$

$$c^x_i, \tag{3.12c}$$

$$c^{ba}_{ji} = c^{ab}_{ij}, \tag{3.12d}$$

$$c^{ba}_{xi}, \tag{3.12e}$$

$$c^{xa}_{ji}, \tag{3.12f}$$

$$c^{yx}_{ji} = -c^{yx}_{ij} = -c^{xy}_{ji} = c^{xy}_{ij}, \tag{3.12g}$$

$$c^{ba}_{yx} = -c^{ba}_{xy} = -c^{ab}_{yx} = c^{ab}_{xy}, \tag{3.12h}$$

and

$$c^{ya}_{jx}. \tag{3.12i}$$

As with the closed-shell case it is easier to project upon a simpler set of states. A possible set is

$$a^\dagger_{a\alpha} a_{i\alpha} |0\rangle, \tag{3.13a}$$

$$a^\dagger_{a\alpha} a_{x\alpha} |0\rangle, \tag{3.13b}$$

$$a^\dagger_{x\beta} a_{i\beta} |0\rangle, \tag{3.13c}$$

$$a^\dagger_{a\sigma_1} a_{i\sigma_1} a^\dagger_{b\sigma_2} a_{j\sigma_2} |0\rangle, \tag{3.13d}$$

$$a^\dagger_{x\beta} a_{i\beta} a^\dagger_{a\sigma} a_{j\sigma} |0\rangle, \tag{3.13e}$$

$$a^\dagger_{a\alpha} a_{x\alpha} a^\dagger_{b\sigma} a_{i\sigma} |0\rangle, \tag{3.13f}$$

$$a^\dagger_{x\beta} a_{i\beta} a^\dagger_{y\beta} a_{j\beta} |0\rangle, \tag{3.13g}$$

$$a^\dagger_{a\alpha} a_{x\alpha} a^\dagger_{b\alpha} a_{y\alpha} |0\rangle, \tag{3.13h}$$

and

$$a^\dagger_{x\beta} a_{i\beta} a^\dagger_{a\alpha} a_{y\alpha} |0\rangle. \tag{3.13i}$$

Note that in these states some of the summations over the spin variable, σ, could not be replaced by an α or β. This is necessary to avoid projecting by states that are linear combinations of states from both inside and outside of the interacting space. The effect of partially including some of the noninteracting space in the projection would typically be an increase in the computed CISD energy, since the noninteracting space can interact through the Hamiltonian operator with only states excited from the reference. However, this is in contrast to those CISD calculations where the noninteracting space is included *in addition* to the interacting space. The extra configurations introduced in this case would always lower the CISD energy, usually by only a small amount.

3.3. High-spin open-shell coupled cluster singles and doubles

The ideal CCSD theory for the HSOS reference would allow direct comparison to the closed-shell CCSD energies and have similar computational resource requirements. Thus, desirable qualities in the HSOS CCSD method would be the use of a reference function optimized for the state under scrutiny and a cluster operator which consisted of all the excitations which can interact with the reference wavefunction through the Hamiltonian. Also, a wavefunction which is an eigenfunction of the S^2 operator is greatly preferred. The ideal HSOS CCSD method would be obtained by substituting the excitation operator, X, introduced in the above HSOS CISD method for the closed-shell cluster operator and substituting the HSOS reference directly for the closed-shell reference. It should be mentioned that since the HSOS reference can be written as a single determinant, the reference can be considered as the quasivacuum. This allows expectation values with respect to the reference to be evaluated directly using Wick's theorem.

The wavefunction in terms of the cluster operator, T, and the reference, $|0\rangle$ is written

$$|\Psi\rangle = e^T|0\rangle, \tag{3.14}$$

where

$$\begin{aligned} T = {} & t_i^a E_i^a|0\rangle + t_x^a E_x^a|0\rangle + t_i^x E_i^x|0\rangle \\ & + t_{ji}^{ba} E_j^b E_i^a|0\rangle + t_{xi}^{ba} E_x^b E_i^a|0\rangle + t_{ji}^{xa} E_j^x E_i^a|0\rangle \\ & + t_{ji}^{yx} E_j^y E_i^x|0\rangle + t_{yx}^{ba} E_y^b E_x^a|0\rangle + t_{jx}^{ya} E_j^y E_x^a|0\rangle. \end{aligned} \tag{3.15}$$

Substituting $|\Psi\rangle$ into the Schrödinger equation gives

$$H_n e^T|0\rangle = E_{\text{corr}} e^T|0\rangle \tag{3.16}$$

and projecting on the left by e^{-T} we obtain

$$H_{\text{eff}}|0\rangle = E_{\text{corr}}|0\rangle, \tag{3.17}$$

where

$$H_{\text{eff}} = e^{-T} H_n e^T. \tag{3.18}$$

The effective Hamiltonian, H_{eff}, can be rewritten as a multicommutator expansion:

$$\begin{aligned} H_{\text{eff}} = {} & 1 + [H, T] + \frac{1}{2}[[H, T], T] + \frac{1}{3!}[[[H, T], T], T] \\ & + \frac{1}{4!}[[[[H, T], T], T], T] + \cdots. \end{aligned} \tag{3.19}$$

For the closed-shell case this expansion terminates after four commutators. Furthermore, for the closed-shell case, the effective Hamiltonian can be written as the connected part of He^T:

$$H_{\text{eff}} = (He^T)_{\text{conn}}. \tag{3.20}$$

Unfortunately, neither of these observations hold in the HSOS CCSD method. The breakdown of these two simplifications occurs because T is made to preserve the spin of any state it acts upon. For example, the term in T which is responsible for single excitations from the open-shell orbitals to the particle orbitals is $t_x^a \boldsymbol{a}_{a\sigma}^\dagger \boldsymbol{a}_{x\sigma}$, which contains an annihilation operator that acts upon open-shell orbitals with β spin. The annihilation of orbitals unoccupied in the reference and the creation of electrons in orbitals already in the reference cause, when quasivacuum expectation values are taken, contractions between the Hamiltonian and/or $\boldsymbol{T}$ operators and $\boldsymbol{T}$ operators to the left of these. The new $\boldsymbol{T}$-$\boldsymbol{H}$ and $\boldsymbol{T}$-$\boldsymbol{T}$ contractions did not exist for the closed-shell case, which only had $\boldsymbol{H}$-$\boldsymbol{T}$ contractions, and are the cause of considerable complexity in the HSOS case. The $\boldsymbol{T}$-$\boldsymbol{H}$ contractions are what make the relation $\boldsymbol{H}_{\text{eff}} = (\boldsymbol{He}^{\boldsymbol{T}})_{\text{conn}}$ invalid, despite the fact that the multicommutator expansion for $\boldsymbol{H}_{\text{eff}}$ makes it a connected quantity.

In addition to the equivalence of the reference and the quasivacuum, which other open-shell coupled cluster cases formulated in this way might not enjoy, the HSOS CCSD case does have another simplification, and that is that the multicommutator expansion does in fact terminate after eight commutators [43] no matter how many singly occupied orbitals there are in the reference. This is because all of the terms in the $\boldsymbol{T}$ operator contain at least one particle or hole index. Particles and holes in the $\boldsymbol{T}$ operator are always created, or, said another way, electrons in particle orbitals are always created and electrons in hole orbitals are always destroyed. Equivalently, the new $\boldsymbol{T}$-$\boldsymbol{H}$ and $\boldsymbol{T}$-$\boldsymbol{T}$ contractions only arise for the open-shell orbitals. Projecting onto doubly excited states can annihilate at most a total of four particle and hole states. The Hamiltonian has at most two-body terms, and therefore, at most, can annihilate four particles and holes. If at most eight total particle and hole excitations can be annihilated, at most eight total particle and hole excitations may be created by the $\boldsymbol{T}$ operators; otherwise the quasivacuum expectation value will be zero. Thus, at most eight $\boldsymbol{T}$ operators may appear in a term in HSOS CCSD theory and the multicommutator expansion must terminate after eight commutators.

The SQSYM program discussed in Sect. 2 has been used to derive the HSOS equations. Care has been taken to make SQSYM an efficient method for deriving second quantization equations. Despite this, some of the derivations are quite demanding. For example, the spin-adapted closed-shell CCSD equations could be derived in a few minutes on a Sun 3/80 workstation. The 202 seconds it took to derive the HSOS triplet terms which were quadratic in $\boldsymbol{T}$ on a MIPS 2000/8 computer, over six times faster than the 3/80, demonstrate the formidable complexity of the equations. Unfortunately, this is just the beginning; the number of terms arising which are cubic in $\boldsymbol{T}$ explodes and the time taken to derive these terms is 3,912 seconds on the MIPS machine. The number of terms quadratic in $\boldsymbol{T}$ is 4,083 and the count of the cubic terms is 12,551. The time needed to evaluate terms with very high powers in $\boldsymbol{T}$ will eventually drop off because SQSYM can recognize which products of $\boldsymbol{T}$ operators cannot make a contribution to the quasivacuum expectation value. The rate limiting step in SQSYM is not the derivation of the equations, however. Most of the processor time is used to compile the equations into the CORR language, for which the SQSYM program required, for all terms up to the cubic power in $\boldsymbol{T}$, 38,279 seconds. Improvements in SQSYM and in computer performance will make the derivation of equations with higher than cubic powers in $\boldsymbol{T}$ possible, but it will be argued in Sect. 4 that these terms will not significantly change the answer for most problems.

The code produced does have the best possible n dependence, where n is the number of electrons. The memory requirement is n^4 and the processor time requirement is n^6 for HSOS CCSD. However, the multiplicative factors for these n dependencies are much larger with the current implementation of SQSYM than they need to be, so current applications of HSOS CCSD theory to actual molecules tend to consume more resources than necessary. Future improvements to SQSYM and the CORR program will relieve some of this computational burden; however, an exhaustive optimization of the equations is much too time consuming, so some manual assistance must be given to SQSYM. One method of providing assistance to SQSYM is to explicitly recognize linear combinations of cluster coefficient which arise naturally in the equations. For the closed-shell case an important improvement in the algorithms is obtained by working with

$$\tau_{ij}^{ab} = t_{ij}^{ab} + t_i^a t_j^b \tag{3.21}$$

instead of just t_{ij}^{ab}. In the HSOS CCSD there is a possibility that great simplifications could be made by working with a τ_{ij}^{ab} expressed in terms of expressions like

$$t_i^a t_j^b, \tag{3.22a}$$

$$t_x^a t_y^b t_i^x t_j^y, \tag{3.22b}$$

$$t_{xy}^{ab} t_{ij}^{xy}, \tag{3.22c}$$

and so on. Similar intermediates might be found for the other two-body cluster coefficients as well, but it is as yet unknown how much benefit the use of these intermediates will provide. Access to faster computers is needed to continue work along these lines.

3.4. *The high-spin open-shell partially spin-adapted* CCSD *method*

The equations for HSOS CCSD theory are quite cumbersome. A simplified approach akin to that used by Nakatsuji and Hirao [29] has also been investigated. Nakatsuji and Hirao recommended using a cluster operator which generates only spin eigenfunctions when it operates upon the reference wavefunction, but removing all operators which give rise to the ***T***-***T*** and ***T***-***H*** contractions, which cause much of the difficulty in HSOS CCSD. This gives a wavefunction which has spin contamination in the terms with two or more powers in the cluster operator, which suggests projecting out the spin contaminants. Nakatsuji and Hirao never implemented this for more than single excitations from the reference. In fact, for triplet states they have employed a completely different method [31], which starts with a reference which is not an eigenstate of $\boldsymbol{S}^2$, perhaps because the complexity of the equations for the method they had earlier proposed prohibited its further development. However, SQSYM has no problem with these equations; in fact, they are quite a bit simpler than the fully spin-adapted HSOS CCSD equations.

The cluster operator in this method, which will be called the high-spin open-shell partially spin-adapted CCSD method (HSOS PSACCSD), is obtained directly from the HSOS CCSD cluster operator by eliminating the components of the cluster operator which annihilate the reference. Thus, for the cluster operator we are left with

$$\begin{aligned} \boldsymbol{T}_{\mathrm{PSA}} = {} & t_i^a \boldsymbol{E}_i^a + t_x^a \boldsymbol{a}_{a\alpha}^\dagger \boldsymbol{a}_{x\alpha} + t_i^x \boldsymbol{a}_{x\beta}^\dagger \boldsymbol{a}_{i\beta} + t_{ji}^{ba} \boldsymbol{E}_j^b \boldsymbol{E}_i^a + t_{xi}^{ba} \boldsymbol{a}_{b\alpha}^\dagger \boldsymbol{a}_{x\alpha} \boldsymbol{E}_i^a + t_{ji}^{xa} \boldsymbol{a}_{x\beta}^\dagger \boldsymbol{a}_{j\beta} \boldsymbol{E}_i^a \\ & + t_{ji}^{yx} \boldsymbol{a}_{y\beta}^\dagger \boldsymbol{a}_{j\beta} \boldsymbol{a}_{x\beta}^\dagger \boldsymbol{a}_{i\beta} + t_{yx}^{ba} \boldsymbol{a}_{b\alpha}^\dagger \boldsymbol{a}_{y\alpha} \boldsymbol{a}_{a\alpha}^\dagger \boldsymbol{a}_{x\alpha} + t_{jx}^{ya} \boldsymbol{a}_{y\beta}^\dagger \boldsymbol{a}_{j\beta} \boldsymbol{a}_{a\alpha}^\dagger \boldsymbol{a}_{x\alpha}. \end{aligned} \tag{3.23}$$

There are some similarities between this choice of the cluster operator and the use of a normal ordered wave operator. The action of the normal ordered HSOS CCSD wave operator, $(e^T)_{\text{n.o.}}$, and the HSOS PSACCSD wave operator, $e^{T_{\text{PSA}}}$, on the reference is identical:

$$(e^T)_{\text{n.o.}}|0\rangle = E^{T_{\text{PSA}}}|0\rangle. \tag{3.24}$$

However, this is only true when the wave operators act upon the reference. In general,

$$((e^T)_{\text{n.o.}})^{-1} \neq e^{-T_{\text{PSA}}} \tag{3.25}$$

and when the inverse of the wave operator is applied to both sides of the Schrödinger equation the theories using the normal ordered wave operator and the exponential wave operator will differ. The advantage of using the exponential wave operator is the simplicity of the form of its inverse.

The equations developed using these operators superficially resemble the closed-shell theory more than the HSOS equations. As with the closed-shell case, H_{eff} terminates at the fourth commutator and $H_{\text{eff}} = (He^T)_{\text{conn}}$. However, one must take care to project out the spin contaminants in the wavefunction by proper choice of the states that are used to project out the equations for the cluster coefficients. The simplified projection scheme used in the HSOS CISD and CCSD methods will produce a spin contaminated wavefunction if used in the HSOS PSACCSD method. Instead, states constructed from products of unitary group generators should be used.

4. Applications of open-shell coupled cluster theory

The techniques discussed in Sect. 2 have been applied to the open-shell coupled cluster theories presented in Sect. 3 as well as the method of Rittby and Bartlett [28] to produce pilot programs that can be used to evaluate the relative potential of these methods. The size of the chemical systems and the basis sets employed are limited to be fairly small, so emphasis will not be placed on comparisons to experiment. Rather, the new methods will be compared to the CISD and the full CI methods.

4.1. The single-triplet splitting in methylene

The energies for the $^3B_1$ and $^1A_1$ states of methylene are reported in Table 6. The configuration interaction with all singles and doubles (CISD), CISD with the Davidson correction (CISD + Dav.), and the full CI results were obtained by Bauschlicher and Taylor [44]. Their CISD calculations did not restrict excitations to the interacting space [45, 46]. This has no effect on the $^1A_1$ energy; however, the $^3B_1$ energy computed in this way yields an energy which is lower than the energy computed with an interacting space CISD. The biggest error in the energy is in the treatment of the $^1A_1$ state, because it is better described with a two reference starting function. Thus, the CISD results fall above the $^1A_1$ energy more than CISD will fall above the $^3B_1$ energy and $\Delta E_{^1A_1-^3B_1}$ will be too large. Not restricting the $^3B_1$ CISD excitations to the first order interacting space will slightly aggravate the problem. Also shown in the table are the results with the $^3B_1$ CISD excitations restricted to the interacting space, CISD(int. space) and

Table 6. The singlet-triplet splitting of methylene, predicted by theoretical methods described in the text. When different methods are used for the singlet and triplet wavefunctions, the notation $^3B_1$ method/$^1A_1$ method is used

Method	$E_{^3B_1}$ (a.u.)	$E_{^1A_1}$ (a.u.)	ΔE (kcal/mol)	Error
SCF	−38.927947	−38.886297	26.14	14.17
CISD	−38.041602	−38.018284	14.63	2.66
CISD + Dav.	−39.046910	−39.027222	12.35	0.38
CISD(int. space)	−39.041199	−39.018284	14.38	2.41
CISD(int. space) + Dav.	−39.046408	−39.027222	12.04	0.07
UCEPA	−39.047715	−39.028718	11.92	−0.05
PCCSD/CCSD(4*c*)	−39.044076	−39.023639	12.82	0.85
PSACCSD(4*c*)/CCSD(4*c*)	−39.043799	−39.023639	12.65	0.68
HSOS CCSD(1*c*)/CCSD(1*c*)	−39.046476	−39.030131	10.26	−1.71
HSOS CCSD(2*c*)/CCSD(2*c*)	−39.044028	−39.023639	12.79	0.82
HSOS CCSD(3*c*)/CCSD(4*c*)	−39.044026	−39.023639	12.79	0.82
full CI	−39.046260	−39.027183	11.97	0

CISD(int. space) + Dav. These show an improvement of $\Delta E_{^1A_1-^3B_1}$ relative to the full CI result by 0.3 kcal/mol. In fact, the CISD(int. space) + Dav. $\Delta E_{^1A_1-^3B_1}$ misses the full CI result by only 0.07 kcal/mol. The CISD + Dav. method frequently gives very good results for small systems, but this method's serendipity will run out as the systems get larger, since CISD + Dav. suffers from CISD's lack of size extensivity.

In all of these CISD calculations only single reference treatments of the $^1A_1$ state have been considered. This is because, although a multireference coupled cluster method which is capable of correctly describing the $^1A_1$ state has already been presented by Paldus et al. [20], we have not yet implemented such a method to compare with the other two reference methods. However, one result is included which uses two references to describe the $^1A_1$ state of methylene and this is the unitary coupled electron pair approximation (UCEPA) method of Hoffmann and Simons [27]. They used a spin-adapted method which produces equations that are linear in an anti-Hermitian cluster operator and which reduces to linearized coupled cluster theory when the reference used is a one configuration closed-shell state. As is commonly the case with linearized CC methods the energies are slightly below the full CI energies. The references these authors employed for their singlet triplet methylene splitting calculations were a single configuration state function (CSF), $1a_1^2 2a_1^2 1b_2^2 3a_1 1b_1$, for the $^3B_1$ state and the two CSF state $c_1 1a_1^2 2a_1^2 1b_2^2 3a_1^2 + c_2 1a_1^2 2a_1^2 1b_2^2 1b_1^2$, for the $^1A_1$ state. It has long been known [47] that a one configuration function for the triplet and a two configuration function for the singlet are needed to give a balanced description of the wavefunctions so that $\Delta E_{^1A_1-^3B_1}$ can be accurately computed without excitations higher than singles and doubles in the wavefunction. The UCEPA method of Hoffmann and Simons gives a quite good $\Delta E_{^1A_1-^3B_1}$.

The acronyms for the single reference coupled cluster methods shown in Table 6 are augmented with a parenthetical suffix giving the maximum number of commutators used to obtain the equations for the energy and wavefunction. Thus, CCSD(1*c*) has only the first commutator implemented and is equivalent to L-CCSD. Starting with the method using HSOS CCSD(1*c*) for the $^3B_1$ state we find that the triplet energy is very close to the full CI energy, but if we use the CCSD(1*c*) method on the $^1A_1$ state to obtain $\Delta E_{^1A_1-^3B_1}$ then the result is quite poor. This is because the single reference CCSD(1*c*) method does a very poor job of describing the inherently two configuration $^1A_1$ state and obtains an energy well below that of the full CI result. However, after we add terms quadratic in $\boldsymbol{T}$ to compute the energy of the $^1A_1$ state with CCSD(2*c*) then the method produces a more reasonable answer and this time gives an answer which is above the full CI answer. Going from CCSD(2*c*) to CCSD(4*c*) ($\equiv$the full CCSD model for a closed-shell reference function) does not change the result by even a μHartree. To obtain $\Delta E_{^1A_1-^3B_1}$ using the CCSD wavefunctions more commutators must be used in the HSOS CCSD method as well. The HSOS CCSD(2*c*) method provides an energy above the full CI energy and adding the third commutator has a negligible effect on the energy. If HSOS CCSD(8*c*) ($\equiv$the full HSOS CCSD method) were currently available, no change in the result would be expected. The splitting $\Delta E_{^1A_1-^3B_1}$ at the HSOS CCSD(3*c*)/CCSD(4*c*) level is quite good compared to the CISD(int. space) results; however, it is clear that a multireference description of the singlet is needed to fully treat this problem with methods based on only single and double excitations.

The PSACCSD method is also compared to CCSD in Table 6. The $\Delta E_{^1A_1-^3B_1}$ is better but this is because PSACCSD overestimates the energy of the triplet state, although the degree of overestimation is quite small. Also provided are results using the PCCSD method of Rittby and Bartlett [28], which produces answers that are very similar to the HSOS CCSD(3*c*) results.

4.2. The $^2B_1$ and $^2A_1$ NH_2 *potential energy surfaces*

The $^2B_1$ and $^2A_1$ states of NH_2 were studied with the open-shell CCSD methods for the equilibrium nuclear geometry as well as geometries where the N–H bond distances have been stretched. Specifically, the geometries employed are r_{eq} ($\equiv$the equilibrium geometry), $1.5r_{eq}$, $2r_{eq}$, and a structure that is essentially dissociated into N atom and H_2. The $^2B_1$ NH_2 results for the open-shell coupled cluster methods are listed in Table 7. All use DZ basis sets and exclude excitations from the lowest lying molecular orbital, the 1*s*-like orbital on nitrogen. What this table shows are the differences between the open-shell coupled cluster methods and the full CI energies which were obtained by Bauschlicher et al. [48], as well as the SCF, CISD, and CISD + Dav. differences, which were obtained by Bauschlicher et al. Their CISD excitations were restricted to the interacting space.

Table 7. Total energies (a.u.) relative to full CI for the $^2B_1$ state of NH_2

Method	ΔE_{r_e}	$\Delta E_{1.5r_e}$	ΔE_{2r_e}	$N + H_2$
SCF	0.102203	0.161029	0.264315	0.089605
CISD	0.004609	0.016439	0.055109	0.006215
CISD + Dav.	0.000447	−0.000890	−0.004487	0.000758
PCCSD	0.001272	0.004273	0.000134	0.002720
PSACCSD(4*c*)	0.001340	0.004566	0.001059	0.002756
HSOS CCSD(1*c*)	0.000281	−0.007639	diverged	−0.011179
HSOS CCSD(2*c*)	0.001342	0.004448	diverged	0.002774
HSOS CCSD(3*c*)	0.001341	0.004487	0.006936	0.002773

Several insights may be gained from such a comparison besides the deviations from the full CI results. The SCF wavefunction becomes a very poor description as the N–H distances are increased; thus this comparison tests the ability of the methods to make up for the deficiencies of a single reference. Also, it is usually more important to obtain correct relative energies, as opposed to absolute energies. If a constant error relative to the full CI is consistently obtained with the method, then for all practical purposes it works as well as full CI for that case.

Examining Table 7 we see the difficulty that SCF has in describing the $^2B_1$ state of NH_2 as the N–H bond distances are increased. The CISD method considerably corrects for this deficiency, but its error is still too large. As is frequently the case for small systems, CISD + Dav. significantly improves upon the CISD energy, giving both smaller and more uniform errors than CISD. However, the CISD + Dav. error drops by 0.0036 Hartree going from $1.5r_{eq}$ to $2r_{eq}$. A similar drop is observed for the PCCSD and PSACCSD methods, although the drop is larger for the PCCSD method. These methods also show a roughly 0.003 Hartree increase going from r_{eq} to $1.5r_{eq}$, which was not observed for the CISD + Dav. method. However, both the PCCSD and PSACCSD methods greatly improve upon the CISD result. The PCCSD results are slightly better than the PSACCSD results using absolute differences from the full CI answers as the yardstick. The HSOS CCSD(3*c*) results are very close to the PSACCSD results for all but the $2r_{eq}$ result, where it is farther from the full CI result than the PSACCSD result. Despite this, it may be stated that the HSOS CCSD(3*c*) answer is better, because it shows a more uniform deviation from the full CI result; but due to difficulties in converging the HSOS CCSD(2*c*) result, it cannot be said that the HSOS CCSD method is converged with respect to the number of commutators for the $2r_{eq}$ energy point.

A similar analysis can be performed on the $^2A_1$ NH_2 results which are displayed in Table 8. These results are similar to the $^2B_1$ NH_2 results except for the structure at $2r_{eq}$, where HSOS CCSD(3*c*) results resemble those for the PCCSD and PSACCSD methods more closely. This does not seem to indicate a change in the behavior for the HSOS CCSD method since its behavior parallels the behavior seen in the $^2B_1$ NH_2 case. However, the PCCSD and PSACCSD methods now exhibit a different trend, instead of giving a drop in energy relative to the full CI energy as the bond lengths increase, the energies now continue increasing as the bond is stretched. This could be interpreted as meaning that the HSOS CCSD is more uniform or stable with respect to changes in the electronic system. In other words, when HSOS CCSD is inappropriate for describing an electronic wavefunction, we have a better chance of predicting in what way the

Table 8. Total energies (a.u.) relative to full CI for the $^2A_1$ state of NH_2

Method	ΔE_{r_e}	$\Delta E_{1.5r_{eq}}$	$\Delta E_{2r_{eq}}$	$N + H_2$
SCF	0.097980	0.138296	0.200654	0.097165
CISD	0.004336	0.012032	0.032600	0.013125
CISD + Dav.	0.000616	0.000893	−0.004761	0.005426
PCCSD	0.001240	0.005042	0.016325	0.007310
PSACCSD(4*c*)	0.001291	0.005240	0.017762	0.007359
HSOS CCSD(1*c*)	0.000408	−0.000675	diverged	diverged
HSOS CCSD(2*c*)	0.001277	0.005070	0.016231	0.007345
HSOS CCSD(3*c*)	0.001274	0.005042	0.017186	0.007351

HSOS CCSD result is incorrect. This hypothesis will have to be confirmed by experience and by including more than three commutators in the equations for the energy and the cluster coefficients. This need for more commutators can be seen by noting that this geometry shows the largest change seen so far for HSOS CCSD in going from two to three commutators. For cases where the SCF reference forms a good first approximation only two commutators are necessary; however, as the SCF description becomes poorer more commutators must be added to get equations with good convergence properties and to obtain an answer that will not change as still more commutators are added.

5. Conclusion

A method for deriving and implementing equations expressed in the second quantization formalism has been developed. This has made possible the rapid development of spin-adapted high-spin open-shell coupled cluster theories and application of these new methods to simple molecular systems. However, the techniques used to produce these results have been kept general. Thus, coupled cluster methods using different reference states can be much more rapidly implemented than would otherwise be possible. Work is planned for implementing a spin-adapted coupled cluster method which starts with an open-shell singlet reference, for example; but the technique is not limited to single reference wavefunctions. The n dependence, where n is the number of electrons, of the processor time requirement and the memory requirement have been reduced to their minimum values in the current implementation. Future implementations need to reduce the polynomial coefficients, which give the memory and processor requirements in terms of powers of n. However, once the SQSYM compiler is able to generate efficient code, then all of the methods that SQSYM has been used to investigate will benefit from this work. Thus, although a general optimization scheme is more complex and time consuming to develop than is the optimization of a single program, an overall savings in development time will be realized for a general compiler capable of optimizing any expression which might arise in electronic structure theory.

All of the high-spin open-shell coupled cluster methods, HSOS CCSD, PSACCSD, and PCCSD, investigated show a significant improvement over the CISD method, even for the small systems investigated here. Which of these methods will become the technique of choice depends upon several factors, including computational efficiency of the energy program, efficiency of the energy gradient with respect to nuclear displacements, and possibly the ease of computing second derivatives of the energy. Ultimately, the PSACCSD and HSOS CCSD methods should use less processor time and memory than the PCCSD method and will probably become competitive with the GUGACI processor requirements. At that time open-shell coupled cluster methods will be as successful in displacing other methods for the description of electron correlation, as has been the case for the closed-shell coupled cluster method.

Acknowledgements. This research was supported by the U.S. National Science Foundation, Grant CHE-8718469. We thank Prof. Gustavo E. Scuseria for helpful discussions.

References

1. Feynman RP (1949) Phys Rev 76:749
2. Goldstone J (1957) Proc Roy Soc (London) A 239:267

3. Hugenholtz NM (1957) Physica 23:481
4. Čížek J (1969) On the use of the cluster expansion and the technique of diagrams in calculations of correlation effects in atoms and molecules, Advances in Chemical Physics 14:35
5. Roothaan CCJ (1951) Rev Mod Phys 23:69
6. Hill TL (1986) An introduction to statistical thermodynamics. Dover, Mineola, N.Y.
7. Coester F, Kümmel H (1960) Nuclear Physics 17:477
8. Sinanoğlu O (1962) J Chem Phys 36:706
9. Thouless DJ (1960) Nuclear Physics 21:225
10. Scuseria GE, Scheiner AC, Lee TJ, Rice JE, Schaefer HF (1987) J Chem Phys 86:2881
11. Besler BH, Scuseria GE, Scheiner AC, Schaefer HF (1988) J Chem Phys 89:360
12. Raghavachari K, Trucks GW, Pople JA, Head-Gordon M (1989) Chem Phys Lett 157:479
13. Bartlett RJ (1989) J Phys Chem 93:1697
14. Mukherjee D, Pal S (1989) Use of cluster expansion methods in the open-shell correlation problem. Advances in Quantum Chemistry 20:291
15. Chaudhuri R, Sinha D, Mukherjee D, On the extensivity of the roots of effective Hamiltonians in many-body formalisms employing incomplete model space (preprint)
16. Jeziorski B, Monkhorst HJ (1981) Phys Rev A 24:1668
17. Laidig WD, Bartlett RJ (1984) Chem Phys Lett 104:424
18. Laidig WD, Saxe P, Bartlett RJ (1987) J Chem Phys 86:887
19. Jeziorski B, Paldus J (1988) J Chem Phys 88:5673
20. Paldus J, Pylypow L, Jeziorski B (1988) in: Kaldor U, (ed) Many-body methods in quantum chemistry, Lecture Notes in Chemistry 52:151. Springer-Verlag, New York Heidelberg Berlin
21. Meissner L, Jankowski K, Wasilewski J (1988) Intern J Quantum Chem 34:535
22. Meissner L, Kucharski SA, Bartlett RJ (1989) J Chem Phys 91:6187
23. Meissner L, Bartlett RJ (1990) J Chem Phys 92:561
24. Banerjee A, Simons J (1981) Intern J Quantum Chem 19:207
25. Baker H, Robb MA (1983) Mol Phys 50:1077
26. Hoffmann MR, Simons J (1988) J Chem Phys 88:993
27. Hoffmann MR, Simons J (1989) J Chem Phys 90:3671
28. Rittby M, Bartlett RJ (1988) J Phys Chem 92:3033
29. Nakatsuji H, Hirao K (1978) J Chem Phys 68:2053
30. Nakatsuji H (1979) Chem Phys Lett 67:329
31. Hirao K, Nakatsuji H (1981) Chem Phys Lett 79:292
32. Jørgensen P, Simons J (1981) Second quantization-based methods in quantum chemistry, Academic Press, New York
33. Szabo A, Ostlund N (1982) Modern quantum chemistry: Introduction to advanced electronic structure theory. Macmillan, New York
34. Wick GC (1950) Phys Rev 80:268
35. Wolfram S (1988) Mathematica. Addison-Wesley, New York
36. Knowles PJ, Handy NC (1988) J Chem Phys 88:6991
37. Wilensky R (1984) LISPcraft. Norten & Company, New York
38. Kernighan BW, Ritchie DM (1988) The C programming language. Prentice Hall, Englewood Cliffs, New Jersey
39. International Business Machines (1988) Parallel Fortran language and library reference, Part Number SC23-0431-0
40. Cole SJ, Purvis GD (1986) Intern J Quantum Chem Symp 20:665
41. Borodin A, Munro I (1975) The computational complexity of algebraic and numeric problems, 45–47. American Elsevier, New York
42. Saxe P, Fox DJ, Schaefer HF, Handy NC (1982) J Chem Phys 77:5584
43. Scuseria GE (1989) (private communication)
44. Bauschlicher CW Jr., Taylor PR (1986) J Chem Phys 85:6510
45. Bunge A (1970) J Chem Phys 53:20
46. McLean AD, Liu B (1973) J Chem Phys 58:1066
47. Bender CF, Schaefer HF, Franceschetti DR, Allen LC (1972) J Am Chem Soc 94:6888
48. Bauschlicher CW Jr., Langhoff SR, Taylor PR, Handy NC, Knowles PJ (1986) J Chem Phys 85:1469