International Tables for Crystallography (2006). Vol. F, Section 25.2.3, pp. 710–716.

protein boundary is poorly defined, the user may specify protein, solvent and excluded volumes, in which case two cutoffs are specified and the intermediate region is marked as neither protein nor solvent.

25.2.2.5.3. *Averaging-mask determination*

If the user does not supply an averaging mask, it is determined by a local correlation method (Vellieux *et al.*, 1995). A large region covering 27 unit cells is selected, and the local correlation between the maps before and after rotation by one of the noncrystallographic symmetry operators is calculated. The largest contiguous region that is in agreement among different NCS operators is isolated from the local correlation map, and a finer local correlation map is calculated over this volume. This process is iterated until a good mask with a detailed boundary is found.

This approach is fully automatic, except in the case where a noncrystallographic symmetry operator intersects a crystallographic symmetry operator, in which case the mask is not uniquely defined, and some user intervention may be required. The method is robust, and by increasing the radius of the sphere within which the local correlation is calculated, it may be used with very poor maps (Cowtan & Main, 1998). The method is easily extended to include information from multiple averaging operators.

25.2.2.5.4. *Fourier transforms*

For simplicity of coding, all Fourier transforms are performed in core using real-to-Hermitian and Hermitian-to-real fast Fourier transforms (FFTs). The data are expanded to space group $P1$ before calculating a map and averaged back to a reciprocal asymmetric unit after inverse transformation. Most of the map modifications preserve crystallographic symmetry, so restricted phases are not constrained except during phase combination.

25.2.2.5.5. *Histogram matching*

The target histograms are calculated from the protein regions of several stationary-atom structures at resolutions from 6 to 1.5 Å, according to the method described by Zhang & Main (1990*a*). The histogram variances should be consistent with the map variances used in scaling the data. The resolution of the target histogram can be accurately matched to the data resolution by averaging the target histograms on either side of the current resolution.

25.2.2.5.6. *Averaging*

Averaging is performed using a single-step approach (Rossmann *et al.*, 1992), in which every copy of the molecule in a 'virtual' asymmetric unit is averaged with every other copy. Density values are obtained at non-grid positions using a 27-point quadratic spectral spline interpolation. A sharpened map is first calculated by dividing by the Fourier transform of the quadratic spline function. The same spline function is then convoluted with the sharpened map to obtain the density value at an arbitrary coordinate (Cowtan & Main, 1998). This approach gives very accurate interpolation from a coarse grid map with relatively little computation and additionally provides gradient information for the refinement of averaging operators.

25.2.2.5.7. *Multi-crystal averaging*

The multi-crystal averaging calculation in *DMMULTI* is equivalent to several single-crystal averaging calculations running simultaneously, with the exception that during the averaging step, the molecule density is averaged across every copy in every crystal form. This average is weighted by the mean figure of merit of each crystal form; this allows the inclusion of unphased crystal forms, since in the first cycle they will have zero weight and therefore not disrupt the phasing that is already present. In subsequent cycles, the unphased form contains phase information from the back-transformed density.

This technique can be extremely useful, since adding a new crystal form usually provides considerably more phase information than adding a new derivative if the cross-rotation and translation functions can be solved.

In the multi-crystal case, averaging is performed using a two-step approach, first building an averaged molecule from all the copies in all crystal forms, then replacing the density in each crystal form with the averaged values. This approach is computationally more efficient when there are many copies of the molecule.

The conceptual flow chart of simultaneous density-modification calculations across multiple crystal forms is shown in Fig. 25.2.2.3(*a*); in practice, this scheme is implemented using a single process and looping over every crystal form at each stage (Fig. 25.2.2.3*b*). Maps are reconstructed from a large data object containing all the reflection data in every crystal form. Averaging is performed using a second data object containing maps of each averaging domain. By this means, an arbitrary number of domains may be averaged across an arbitrary number of crystal forms.

Multi-crystal averaging has been particularly successful in solving structures from very weak initial phasing, since the data redundancy is usually higher than for single-crystal problems.

25.2.3. The structure-determination language of the *Crystallography & NMR System* (A. T. Brunger, P. D. Adams, W. L. DeLano, P. Gros, R. W. Grosse-Kunstleve, J.-S. Jiang, N. S. Pannu, R. J. Read, L. M. Rice and T. Simonson)

25.2.3.1. *Introduction*

We have developed a new and advanced software system, the *Crystallography & NMR System* (CNS), for crystallographic and NMR structure determination (Brünger *et al.*, 1998). The goals of CNS are: (1) to create a flexible computational framework for exploration of new approaches to structure determination; (2) to provide tools for structure solution of difficult or large structures; (3) to develop models for analysing structural and dynamical properties of macromolecules; and (4) to integrate all sources of information into all stages of the structure-determination process.

To meet these goals, algorithms were moved from the source code into a symbolic structure-determination language which represents a new concept in computational crystallography. The high-level CNS computing language allows definition of symbolic target functions, data structures, procedures and modules. The CNS program acts as an interpreter for the high-level CNS language and includes hard-wired functions for efficient processing of computing-intensive tasks. Methods and algorithms are therefore more clearly defined and easier to adapt to new and challenging problems. The result is a multi-level system which provides maximum flexibility to the user (Fig. 25.2.3.1). The CNS language provides a common framework for nearly all computational procedures of structure determination. A comprehensive set of crystallographic procedures for phasing, density modification and refinement has been implemented in this language. Task-oriented input files written in the CNS language, which can also be accessed through an HTML graphical interface (Graham, 1995), are available to carry out these procedures.

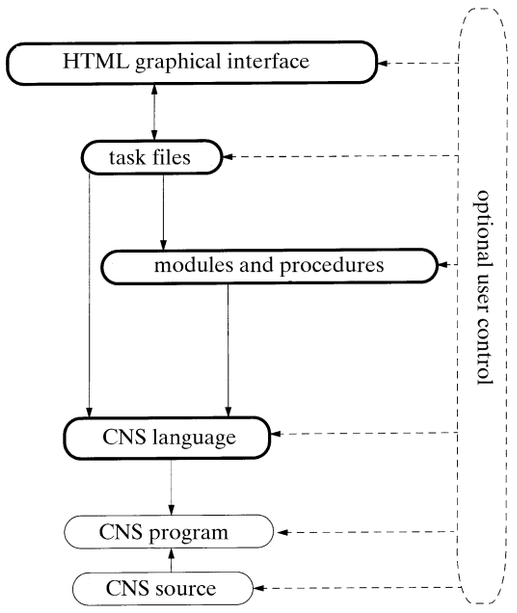

Fig. 25.2.3.1. CNS consists of five layers which are under user control. The high-level HTML graphical interface interacts with the task-oriented input files. The task files use the CNS language and the modules. The modules contain CNS language statements. The CNS language is interpreted by the CNS Fortran77 program. The program performs the data manipulations, data operations and 'hard-wired' algorithms.

25.2.3.2. *The CNS language*

One of the key features of the CNS language is symbolic data structure manipulation, for example,

```
xray
  do (pa = −2 ∗ (amplitude(fp)^2 + amplitude(fh)^2
          − amplitude(fph)^2) ∗ amplitude(fp)
          ∗ real(fh)/(3 ∗ v^2 + 4 ∗ (amplitude(fph)^2
          + sph^2) ∗ v)) (acentric)
end
```
(25.2.3.1)

which is equivalent to the following mathematical expression for all acentric indices **h**,

$$p_a(\mathbf{h}) = 2\frac{-[|\mathbf{f}_p(\mathbf{h})|^2 + |\mathbf{f}_h(\mathbf{h})|^2 - |\mathbf{f}_{ph}(\mathbf{h})|^2]|\mathbf{f}_p(\mathbf{h})|\{[\mathbf{f}_h(\mathbf{h}) + \mathbf{f}_h(\mathbf{h})^*]/2\}}{3v(\mathbf{h})^2 + 4[|\mathbf{f}_{ph}(\mathbf{h})|^2 + s_{ph}(\mathbf{h})^2]v(\mathbf{h})},$$

(25.2.3.2)

where $\mathbf{f}_p$ ['fp' in equation (25.2.3.1)] is the 'native' structure-factor array, $\mathbf{f}_{ph}$ ['fph' in equation (25.2.3.1)] is the derivative structure-factor array, s_{ph} ['sph' in equation (25.2.3.1)] is the corresponding experimental σ, v is the expectation value for the lack of closure (including lack of isomorphism and errors in the heavy-atom model), and $\mathbf{f}_h$ ['fh' in equation (25.2.3.1)] is the calculated heavy-atom structure-factor array. This expression computes the $A_{\rm iso}$ coefficient of the phase probability distribution for single isomorphous replacement described by Hendrickson & Lattman (1970) and Blundell & Johnson (1976).

The expression in equation (25.2.3.1) is computed for the specified subset of reflections '(acentric)'. This expression means that only the selected (in this case all acentric) reflections are used. More sophisticated selections are possible, *e.g.*

```
(amplitude(fp) > 2 ∗ sh and amplitude(fph) > 2 ∗ sph
  and d >= 3)
```
(25.2.3.3)

selects all reflections with Bragg spacing, d, greater than 3 Å for which both native (fp) and derivative (fph) amplitudes are greater than two times their corresponding σ values ('sh' and 'sph', respectively). Extensive use of this structure-factor selection facility is made for cross-validating statistical properties, such as R values (Brünger, 1992*b*), σ_A values (Kleywegt & Brünger, 1996; Read, 1997) and maximum-likelihood functions (Pannu & Read, 1996*a*; Adams *et al.*, 1997).

Similar operations exist for electron-density maps, *e.g.*

```
xray
  do (map = 0) (map < 0.1)
end
```
(25.2.3.4)

is an example of a truncation operation: all map values less than 0.1 are set to 0. Atoms can be selected based on a number of atomic properties and descriptors, *e.g.*

```
do (b = 10)(residue 1:40 and
        (name ca or name n or name c or name o))
```
(25.2.3.5)

sets the B factors of all polypeptide backbone atoms of residues 1 through 40 to 10 Å^2.

Operations exist between data structures, *e.g.* real- and reciprocal-space arrays, and atom properties. For example, Fourier transformations between real and reciprocal space can be accomplished by the following CNS commands:

```
xray
  mapresolution infinity 3.
  fft grid 0.3333 end
  do (map = ft(f_cal)) (acentric)
end
```
(25.2.3.6)

which computes a map on a 1 Å grid by Fourier transformation of the 'f_cal' array for all acentric reflections.

Atoms can be associated with calculated structure factors, *e.g.*

```
associate f_cal (residue 1:50)
```
(25.2.3.7)

This statement will associate the reciprocal-space array 'f_cal' with the atoms belonging to residues 1 through 50. These structure-factor associations are used in the symbolic target functions described below.

There are no predefined reciprocal- or real-space arrays in CNS. Dynamic memory allocation allows one to carry out operations on arbitrarily large data sets with many individual entries (*e.g.* derivative diffraction data) without the need to recompile the source code. The various reciprocal-space structure-factor arrays must therefore be declared and their type specified prior to invoking them. For example, a reciprocal-space array with real values, such as observed amplitudes, is declared by

```
declare name = fobs type = real domain = reciprocal end
```
(25.2.3.8)

Reciprocal-space arrays can be grouped. For example, Hendrickson & Lattman (1970) coefficients are represented as a group of four reciprocal-space structure-factor arrays,

```
evaluate ( $crystal_lattice.space_group     = "P2(1)2(1)2(1)" )
evaluate ( $crystal_lattice.unit_cell.a     = 61.76           )
evaluate ( $crystal_lattice.unit_cell.b     = 40.73           )
evaluate ( $crystal_lattice.unit_cell.c     = 26.74           )
evaluate ( $crystal_lattice.unit_cell.alpha = 90              )
evaluate ( $crystal_lattice.unit_cell.beta  = 90              )
evaluate ( $crystal_lattice.unit_cell.gamma = 90              )
```

(*a*)

```
define (
   &crystal_lattice.space_group     = P2(1)2(1)2(1) ;
   &crystal_lattice.unit_cell.a     = 61.76 ;
   &crystal_lattice.unit_cell.b     = 40.73 ;
   &crystal_lattice.unit_cell.c     = 26.74 ;
   &crystal_lattice.unit_cell.alpha = 90 ;
   &crystal_lattice.unit_cell.beta  = 90 ;
   &crystal_lattice.unit_cell.gamma = 90 ;
)
```

(*b*)

Fig. 25.2.3.2. Examples of compound symbols and compound parameters. (*a*) The 'evaluate' statement is used to define typed symbols (strings, numbers and logicals). Symbol names are in bold. (*b*) The 'define' statement is used to define untyped parameters. Each parameter entry is terminated by a semicolon. The compound base name 'crystal_lattice' has a number of sub-levels, such as 'space_group' and the 'unit_cell' parameters. 'unit_cell' is itself base to a number of sub-levels, such as 'a' and 'alpha'. Parameter names are in bold.

```
group type = hl object = pa object = pb
   object = pc object = pd end                    (25.2.3.9)
```

where 'pa', 'pb', 'pc' and 'pd' refer to the individual arrays. This group statement indicates to CNS that the specified arrays need to be transformed together when reflection indices are changed, *e.g.* during expansion of the diffraction data to space group *P*1.

25.2.3.3. *Symbols and parameters*

The CNS language supports two types of data elements which may be used to store and retrieve information. *Symbols* are typed variables, such as numbers, character strings of restricted length and logicals. *Parameters* are untyped data elements of arbitrary length that may contain collections of CNS commands, numbers, strings, or symbols.

Symbols are denoted by a dollar sign ($), and parameters by an ampersand (&). Symbols and parameters may contain a single data element, or they may be a *compound* data structure of arbitrary complexity. The hierarchy of these data structures is denoted using a period (.). Figs. 25.2.3.2(*a*) and (*b*) demonstrate how crystal-lattice information can be stored in compound symbols and parameters, respectively. The information stored in symbols or parameters can be retrieved by simply referring to them within a CNS command: the symbol or parameter name is substituted by its content. Symbol substitution of portions of the compound names (*e.g.* '&crystal_lattice.unit_cell.$para') allows one to carry out conditional and iterative operations on data structures, such as matrix multiplication.

25.2.3.4. *Statistical functions*

The CNS language contains a number of statistical operations, such as binwise averages and summations. The resolution bins are defined by a central facility in CNS.

Fig. 25.2.3.3 shows how σ_A, σ_Δ and D (Read, 1986, 1990) are computed from the observed structure factors ('fobs') and the calculated model structure factors ('fcalc') using the CNS statistical operations. The first five operations are performed for the reflections in the test set, while the last three operations expand the results to all reflections. The 'norm' function computes normalized structure-factor amplitudes for the specified arguments. The 'sigacv' function

```
do (eobs=norm(amplitude(fobs)))                              ( test_set )
do (ecalc=norm(amplitude(fcalc)))                            ( test_set )
do (sigmaA=sigacv(eobs,ecalc))                               ( test_set )
do (sigmaD=sqrt(save(amplitude(fobs))^2 (1-sigmaA^2)))       ( test_set )
do (D= sigmaA * sqrt(save(amplitude(fobs))^2  /
          save(amplitude(fcalc)^2)))                         ( test_set )

do (sigmaA=sum(sigmaA*test)/ max(1,sum(test)))               ( all )
do (sigmaD=sum(sigmaD*test)/ max(1,sum(test)))               ( all )
do (D=sum(D*test)/ max(1,sum(test)))                         ( all )
```

Fig. 25.2.3.3. Example for statistical operations provided by the CNS language. 'norm', 'sigacv', 'save' and 'sum' are functions that are computed internally by the CNS program. Binwise operations are in italics ('sigacv', 'save' and 'sum'). The result for a particular bin is stored in all elements belonging to the bin. The σ_A ('sigmaA') parameters are computed in binwise resolution shells. The σ_Δ ('sigmaD') and D parameters are then computed from σ_A and binwise averages involving $|\mathbf{F}_o|^2$ and $|\mathbf{F}_c|^2$. The binwise results are expanded to all reflections by the last three statements. 'test' is an array that is 1 for all reflections in the test set and 0 otherwise. 'sum' is a binwise operation on all reflections with the same partitioning used for the test set.

evaluates σ_A from the normalized structure factors. The 'save' function computes the statistical average

$$\mathrm{save}(f) = \frac{\sum_{hkl} f_{hkl}(w/\varepsilon)}{\sum_{hkl} w}, \qquad (25.2.3.10)$$

where w is 1 and 2 for centric and acentric reflections, respectively, and ε is the statistical weight. The averages are computed binwise, and the result for a particular bin is stored in all selected reflections belonging to the bin.

25.2.3.5. *Symbolic target function*

One of the key innovative features of CNS is the ability to symbolically define target functions and their first derivatives for crystallographic searches and refinement. This allows one conveniently to implement new crystallographic methodologies as they are being developed.

The power of symbolic target functions is illustrated by two examples. In the first example, a target function is defined for simultaneous heavy-atom parameter refinement of three derivatives. The sites for each of the three derivatives can be disjoint or identical, depending on the particular situation. For simplicity, the Blow & Crick (1959) approach is used, although maximum-likelihood targets are also possible (see below). The heavy-atom sites are refined against the target

$$\sum_{hkl} \frac{(|\mathbf{F}_{h_1} + \mathbf{F}_p| - |\mathbf{F}_{ph_1}|)^2}{2v_1} + \frac{(|\mathbf{F}_{h_2} + \mathbf{F}_p| - |\mathbf{F}_{ph_2}|)^2}{2v_2} + \frac{(|\mathbf{F}_{h_3} + \mathbf{F}_p| - |\mathbf{F}_{ph_3}|)^2}{2v_3}. \qquad (25.2.3.11)$$

$\mathbf{F}_{h_1}$, $\mathbf{F}_{h_2}$ and $\mathbf{F}_{h_3}$ are complex structure factors corresponding to the three sets of heavy-atom sites, $\mathbf{F}_p$ represents the structure factors of the native crystal, $|\mathbf{F}_{ph_1}|$, $|\mathbf{F}_{ph_2}|$ and $|\mathbf{F}_{ph_3}|$ are the structure-factor amplitudes of the derivatives, and v_1, v_2 and v_3 are the variances of the three lack-of-closure expressions. The corresponding target expression and its first derivatives with respect to the calculated structure factors are shown in Fig. 25.2.3.4(*a*). The derivatives of the target function with respect to each of the three associated structure-factor arrays are specified with the 'dtarget' expressions. The 'tselection' statement specifies the selected subset of reflections to be used in the target function (*e.g.* excluding outliers), and the 'cvselection' statement specifies a subset of reflections to be used for cross-validation (Brünger, 1992*b*) (*i.e.* the subset is not used

```
associate f_h_1 <atom-selection-1>
associate f_h_2 <atom-selection-2>
associate f_h_3 <atom-selection-3>

target=(
         (abs(f_h_1+f_p)-f_ph_1)^2 / (2*v_1) +
         (abs(f_h_2+f_p)-f_ph_2)^2 / (2*v_2) +
         (abs(f_h_3+f_p)-f_ph_3)^2 / (2*v_3)
       )

dtarget(f_h_1)=
        (
         2*(abs(f_h_1+f_p)-f_ph_1)
           *(f_h_1+f_p)/abs(f_h_1+f_p) / (2*v_1)
         )

dtarget(f_h_2)=
        (
         2*(abs(f_h_2+f_p)-f_ph_2)
           *(f_h_2+f_p)/abs(f_h_2+f_p) / (2*v_2)
         )

dtarget(f_h_3)=
        (
         2*(abs(f_h_3+f_p)-f_ph_3)
           *(f_h_3+f_p)/abs(f_h_3+f_p) / (2*v_3)

tselection=<selection>
cvselection=<selection>
```

(*a*)

```
associate fcalc1 <atom-selection1>
associate fcalc2 <atom-selection2>

target=( abs(fobs) - sqrt(abs(fcalc1)^2+abs(fcalc2)^2))^2 )

dtarget(fcalc1)=( -2 *
            (abs(fobs)-sqrt(abs(fcalc1)^2+abs(fcalc2)^2)) *
              (fcalc1/(sqrt(abs(fcalc1)^2+abs(fcalc2)^2))))

dtarget(fcalc2)=( -2 *
            (abs(fobs)-sqrt(abs(fcalc1)^2+abs(fcalc2)^2)) *
              (fcalc2/(sqrt(abs(fcalc1)^2+abs(fcalc2)^2))))

tselection=<selection>
cvselection=<selection>
```

(*b*)

Fig. 25.2.3.4. Examples of symbolic definition of a refinement target function and its derivatives with respect to the calculated structure-factor arrays. (*a*) Simultaneous refinement of heavy-atom sites of three derivatives. The target function is defined by the 'target' expression. 'f_h_1', 'f_h_2' and 'f_h_3' (in bold) are complex structure factors corresponding to three sets of heavy atoms that are specified using atom selections [equation (25.2.3.7)]. The target function and its derivatives with respect to the three structure-factor arrays are defined symbolically using the structure-factor amplitudes of the native crystal, 'f_p', those of the derivatives, 'f_ph_1', 'f_ph_2', 'f_ph_3', the complex structure factors of the heavy-atom models, 'f_h_1', 'f_h_2', 'f_h_3', and the corresponding lack-of-closure variances, 'v_1', 'v_2' and 'v_3'. The summation over the selected stucture factors ('tselection') is performed implicitly. (*b*) Refinement of two independent models against perfectly twinned data. 'fcalc1' and 'fcalc2' are complex structure factors for the models that are related by a twinning operation (in bold). The target function and its derivatives with respect to the two structure-factor arrays are explicitly defined.

during refinement but only as a monitor for the progress of refinement).

The second example is the refinement of a perfectly twinned crystal with overlapping reflections from two independent crystal lattices. Refinement of the model is carried out against the residual

$$\sum_{hkl} |\mathbf{F}_{\rm obs}| - (|\mathbf{F}_{\rm calc1}|^2 + |\mathbf{F}_{\rm calc2}|^2)^{1/2}. \qquad (25.2.3.12)$$

The symbolic definition of this target is shown in Fig. 25.2.3.4(*b*). The twinning operation itself is imposed as a relationship between the two sets of selected atoms (not shown). This example assumes that the two calculated structure-factor arrays ('fcalc1' and 'fcalc2') that correspond to the two lattices have been appropriately scaled with respect to the observed structure factors, and the twinning fractions have been incorporated into the scale factors. However, a more sophisticated target function could be defined which incorporates scaling.

A major advantage of the symbolic definition of the target function and its derivatives is that any arbitrary function of structure-factor arrays can be used. This means that the scope of possible targets is not limited to least-squares targets. Symbolic definition of numerical integration over unknown variables (such as phase angles) is also possible. Thus, even complicated maximum-likelihood target functions (Bricogne, 1984; Otwinowski, 1991; Pannu & Read, 1996*a*; Pannu *et al.*, 1998) can be defined using the CNS language. This is particularly valuable at the prototype stage. For greater efficiency, the standard maximum-likelihood targets are provided through CNS source code which can be accessed as functions in the CNS language. For example, the maximum-likelihood target function MLF (Pannu & Read, 1996*a*) and its derivative with respect to the calculated structure factors are defined as

```
target  = (mlf (fobs, sigma, (fcalc + fbulk),
                d, sigma_delta))
dtarget = (dmlf (fobs, sigma, (fcalc + fbulk),
                d, sigma_delta))                    (25.2.3.13)
```

where 'mlf( )' and 'dmlf( )' refer to internal maximum-likelihood functions, 'fobs' and 'sigma' are the observed structure-factor amplitudes and corresponding σ values, 'fcalc' is the (complex) calculated structure-factor array, 'fbulk' is the structure-factor array for a bulk solvent model, and 'd' and 'sigma_delta' are the cross-validated D and σ_Δ functions (Read, 1990; Kleywegt & Brünger, 1996; Read, 1997) which are precomputed prior to invoking the MLF target function using the test set of reflections. The availability of internal Fortran subroutines for the most computing-intensive target functions and the symbolic definitions involving structure-factor arrays allow for maximal flexibility and efficiency. Other examples of available maximum-likelihood target functions include MLI (intensity-based maximum-likelihood refinement), MLHL [crystallographic model refinement with prior phase information (Pannu *et al.*, 1998)], and maximum-likelihood heavy-atom parameter refinement for multiple isomorphous replacement (Otwinowski, 1991) and MAD phasing (Hendrickson, 1991; Burling *et al.*, 1996). Work is in progress to define target functions that include correlations between different heavy-atom derivatives (Read, 1994).

```
module { compute_unit_cell_volume }
(
   &cell;
   &volume;
)


evaluate ( $cabg.1=cos(&cell.alpha) )
evaluate ( $sabg.1=sin(&cell.alpha) )

evaluate ( $cabg.2=cos(&cell.beta) )
evaluate ( $sabg.2=sin(&cell.beta) )

evaluate ( $cabg.3=cos(&cell.gamma) )
evaluate ( $sabg.3=sin(&cell.gamma) )

evaluate ( &volume=&cell.a * &cell.b * &cell.c *
                   sqrt(1+2*$cabg.1*$cabg.2*$cabg.3
                         -$cabg.1^2-$cabg.2^2-$cabg.3^2) )
```

Fig. 25.2.3.5. Use of compound parameters within a module. This module computes the unit-cell volume (Stout & Jensen, 1989) from the unit-cell geometry. Input and output parameter base names are in bold. Local symbols, such as cabg.1, are defined through 'evaluate' statements. The result is stored in the parameter '&volume' which is passed to the invoking task file or module.

```
module { phase_distribution }
     (
   &fp;  {input: native data}
   &sp;  {input: native sigma}
   &sel; {input: selection of structure factors}
   &fh;  {input: name of heavy atom structure factors}
   &fph; {input: name of derivative data array}
   &sph; {input: name of derivative's sigma array}
   &var; {input: lack-of-isomorphism plus measurement errors}
   &pa;  {output: Hendrickson and Lattman A array}
   &pb;  {output: Hendrickson and Lattman B array}
   &pc;  {output: Hendrickson and Lattman C array}
   &pd;  {output: Hendrickson and Lattman D array}
     )

do (&pa= (cos(centric_phase)*(
       -(abs(&fh+combine(abs(&fp),centric_phase))-abs(&fph))^2/(2*&var)
       +(abs(&fh-combine(abs(&fp),centric_phase))-abs(&fph))^2/(2*&var)))
  ( centric and &sel )

do (&pb=(sin(centric_phase)*(
       -(abs(&fh+combine(abs(&fp),centric_phase))-abs(&fph))^2/(2*&var)
       +(abs(&fh-combine(abs(&fp),centric_phase))-abs(&fph))^2/(2*&var)))
  ( centric and &sel )

do (&pc=0) (centric and &sel)

do (&pd=0) (centric and &sel)

do (&pa=-2 * (amplitude(&fp)^2 + amplitude(&fh)^2 - amplitude(&fph)^2 )
           * amplitude(&fp) * real(&fh)/
           (3 * &var^2 + 4 * (amplitude(&fph)^2+&sph^2) * &var))
  ( acentric and &sel )

do (&pb=-2 * (amplitude(&fp)^2 + amplitude(&fh)^2 - amplitude(&fph)^2 )
           * amplitude(&fp) * imag(&fh)/
           (3 * &var^2 + 4 * (amplitude(&fph)^2+&sph^2) * &var) )
  ( acentric and &sel )

do (&pc=-amplitude(&fp)^2 * (real(&fh)^2 - imag(&fh)^2) /
         (3 * &var^2 + 4 * (amplitude(&fph)^2+&sph^2) * &var) )
  ( acentric and &sel )

do (&pd=-2 * amplitude(&fp)^2 * real(&fh) * imag(&fh) /
        (3 * &var^2 + 4 * (amplitude(&fph)^2+&sph^2) * &var) )
  ( acentric and &sel )
```

(*a*)

```
@phase_distribution
   (
    &fp=fobs;
    &sp=sigma;
    &sel=( d > 3. );
    &fh=f_heavy;
    &fph=f_deriv;
    &sph=s_deriv;
    &var=variance;
    &pa=pa;
    &pb=pb;
    &pc=pc;
    &pd=pd;
   )
```

(*b*)

Fig. 25.2.3.6. Example of (*a*) a CNS module and (*b*) the corresponding module invocation. Input and output parameters are in bold. The module invocation is performed by specifying the '@' character, followed by the name of the module file and the module parameter substitutions. The ampersand (&) indicates that the particular symbol (*e.g.* '&fp') is substituted with the specified value in the invocation statement [*e.g.* 'fobs' in the case of '&fp' in (*b*)]. The module parameter substitution is performed literally, and any string of characters between the equal sign and the semicolon will be substituted.

25.2.3.6. *Modules and procedures*

Modules exist as separate files and contain collections of CNS commands related to a particular task. In contrast, *procedures* can be defined and invoked from within any file. Modules and procedures share a similar parameter-passing mechanism for both input and output. Modules and procedures make it possible to write programs in the CNS language in a manner similar to that of a computing language, such as Fortran or C. CNS modules and procedures have defined sets of input (and output) parameters that are passed into them (or returned) when they are invoked. This enables long collections of CNS language statements to be broken down into modules for greater clarity of the underlying algorithm.

Parameters passed into a module or procedure inherit the scope of the calling task file or module, and thus they exhibit a behaviour analogous to most computing languages. Symbols defined within a module or procedure are purely local variables.

Experimental phasing
- Heavy-atom (Patterson) searches
- Patterson refinement
- Multiple isomorphous replacement phasing and site refinement
- Multi-wavelength anomalous dispersion phasing and site refinement

Molecular replacement
- Patterson real-space and direct rotation searches
- Patterson-correlation refinement
- Fast FFT translation search

Density modification
- Creation of envelopes
- Solvent flattening
- Density averaging
- Histogram matching

Refinement
- Maximum-likelihood targets
- Torsion-angle molecular dynamics
- Cartesian molecular dynamics
- Conjugate-gradient minimization
- Composite annealed omit map

Other
- Protein Data Bank deposition file generation
- mmCIF file creation

Fig. 25.2.3.7. Procedures and features available in CNS for structure determination by X-ray crystallography.

The following example shows how the unit-cell parameters defined above (Fig. 25.2.3.2*b*) are passed into a module named 'compute_unit_cell_volume' (Fig. 25.2.3.5), which computes the volume of the unit cell from the crystal lattice parameters using well established formulae (Stout & Jensen, 1989):

```
@compute_unit_cell_volume (cell = &crystal_lattice.unit_cell;
                           volume = $cell_volume;)
```

(25.2.3.14)

The parameter 'volume' is equated to the symbol '$cell_volume' upon invocation in order to return the result (the unit-cell volume) from this module. Note that the use of compound parameters to define the crystal lattice parameters (Fig. 25.2.3.2*b*) provides a convenient way to pass all required information into the module by referring to the base name of the compound parameter ('&crystal_lattice.unit_cell') instead of having to specify each individual data element.

Fig. 25.2.3.6(*a*) shows another example of a CNS module: the module named 'phase_distribution' computes phase probability distributions using the Hendrickson & Lattman formalism (Hendrickson & Lattman, 1970; Hendrickson, 1979; Blundell & Johnson, 1976). An example for invoking the module is shown in Fig. 25.2.3.6(*b*). This module could be called from task files that need access to isomorphous phase probability distributions. It would be straightforward to change the module in order to compute different expressions for the phase probability distributions.

A large number of additional modules are available for crystallographic phasing and refinement. CNS library modules include space-group information, Gaussian atomic form factors, anomalous-scattering components, and molecular parameter and topology databases.

```
{+ file: anneal.inp +}
{+ description: Crystallographic simulated annealing refinement +}
{+ authors: Axel T. Brunger, Luke M. Rice and Paul D. Adams +}
{+ reference: A.T. Brunger, J. Kuriyan and M. Karplus, Crystallographic
              R factor Refinement by Molecular Dynamics, Science
              235, 458-460 (1987) +}
{+ reference: A.T. Brunger, A. Krukowski and J. Erickson, Slow-Cooling
              Protocols for Crystallographic Refinement by Simulated
              Annealing, Acta Cryst. A46, 585-593 (1990) +}
{- begin block parameter definition -} define(
{====================== crystallographic data ========================}
{* space group *}
{* use International Table conventions with subscripts substituted by
   parenthesis *}
{===>} sg="P2(1)2(1)2(1)";

{* unit cell *}
{===>} a=61.76;   {===>} b=40.73;  {===>} c=26.74;
{===>} alpha=90;  {===>} beta=90;  {===>} gamma=90;

{* anomalous f' f'' library file *}
{* should be used when refining against anomalous data -
   libraries: "CNS_XTALLIB:anom_cu.lib" and "CNS_XTALLIB:anom_mo.lib" or
              a user created file.
   If blank no anomalous contribution will be included in the refinement *}
{===>} anom_library="";

{* reflection file *}
{===>} ref="example.hkl";

{* reciprocal space array containing observed amplitudes: required *}
{===>} obs_f="f_native";

{* reciprocal space array containing sigma values for amplitudes: required *}
{===>} obs_sigf="s_native";

{* reciprocal space array containing test set for cross-validation: required *}
{===>} test_set="test";
{* refinement target *}
{* mlf: maximum likelihood target using amplitudes
   mli: maximum likelihood target using intensities
   mlhl: maximum likelihood target using amplitudes and phase probability
         distribution
   residual: standard crystallographic residual
   vector: vector residual
   mixed: (1-fom)*residual + fom*vector
   e2e2: correlation coefficient using normalized E^2
   e1e1: correlation coefficient using normalized E
   f2f2: correlation coefficient using F^2
   f1f1: correlation coefficient using F *}
{+ choice: "mlf" "mli" "mlhl" "residual" "vector" "mixed"
           "e2e2" "e1e1" "f2f2" "f1f1" +}
{===>} reftarget="mlf";
}  {- end block parameter definition -}
```

Fig. 25.2.3.8. Example of a typical CNS task file: a section of the top portion of the simulated-annealing refinement protocol which contains the definition of various parameters that are needed in the main body of the task file. Each parameter is indicated by a name, an equal sign and an arbitrary sequence of characters terminated by a semicolon (*e.g.* 'a = 61.76;'). The top portion of each task file also contains commands for the HTML interface embedded in comment fields (indicated by braces, {...}). The commands that can be modified by the user in the HTML form are in bold.

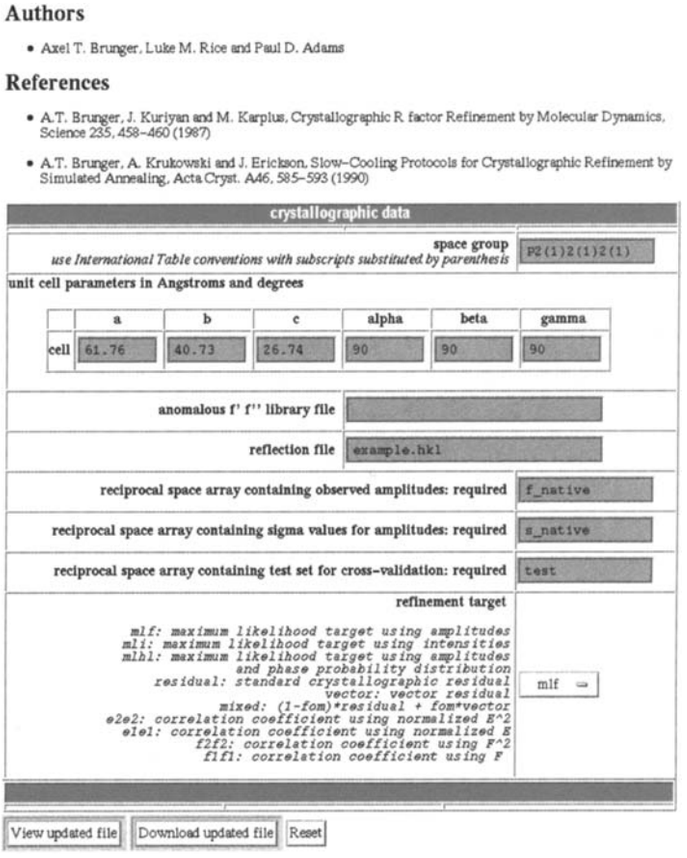

Fig. 25.2.3.9. Example of a CNS HTML form page. This particular example corresponds to the task file in Fig. 25.2.3.8.

25.2.3.7. *Task files*

Task files consist of CNS language statements and module invocations. The CNS language permits the design and execution of nearly any numerical task in X-ray crystallographic structure determination using a minimal set of 'hard-wired' functions and routines. A list of the currently available crystallographic procedures and features is shown in Fig. 25.2.3.7.

Each task file is divided into two main sections: the initial parameter definition and the main body of the task file. The definition section contains definitions of all CNS parameters that are used in the main body of the task file. Modification of the main body of the file is not required, but may be done by experienced users in order to experiment with new algorithms. The definition section also contains the directives that specify specific HTML features, *e.g.* text comments (indicated by {*...*}), user-modifiable fields (indicated by {===>}), and choice boxes (indicated by {+ choice: ...+}). Fig. 25.2.3.8 shows a portion of the 'define' section of a typical CNS refinement task file.

The task files produce a number of output files (*e.g.* coordinate, reflection, graphing and analysis files). Comprehensive information about input parameters and results of the task are provided in these output files. In this way, the majority of the information required to reproduce the structure determination is kept with the results. Analysis data are often given in simple columns and rows of numbers. These data files can be used for graphing, for example, by using commonly available spreadsheet programs. An HTML graphical output feature for CNS which makes use of these analysis files is planned. In addition, list files are often produced that contain a synopsis of the calculation.

25.2.3.8. *HTML interface*

The HTML graphical interface uses HTML to create a high-level menu-driven environment for CNS (Fig. 25.2.3.9). Compact and relatively simple Common Gateway Interface (CGI) conversion scripts are available that transform a task file into a form page and the edited form page back into a task file (Fig. 25.2.3.10). These conversion scripts are written in PERL.

A comprehensive collection of task files are available for crystallographic phasing and refinement (Fig. 25.2.3.7). New task files can be created or existing ones modified in order to address problems that are not currently met by the distributed collection of task files. The HTML graphical interface thus provides a common interface for distributed and 'personal' CNS task files (Fig. 25.2.3.10).

25.2.3.9. *Example: combined maximum-likelihood and simulated-annealing refinement*

CNS has a comprehensive task file for simulated-annealing refinement of crystal structures using Cartesian (Brünger *et al.*, 1987; Brünger, 1988) or torsion-angle molecular dynamics (Rice & Brünger, 1994). This task file automatically computes cross-

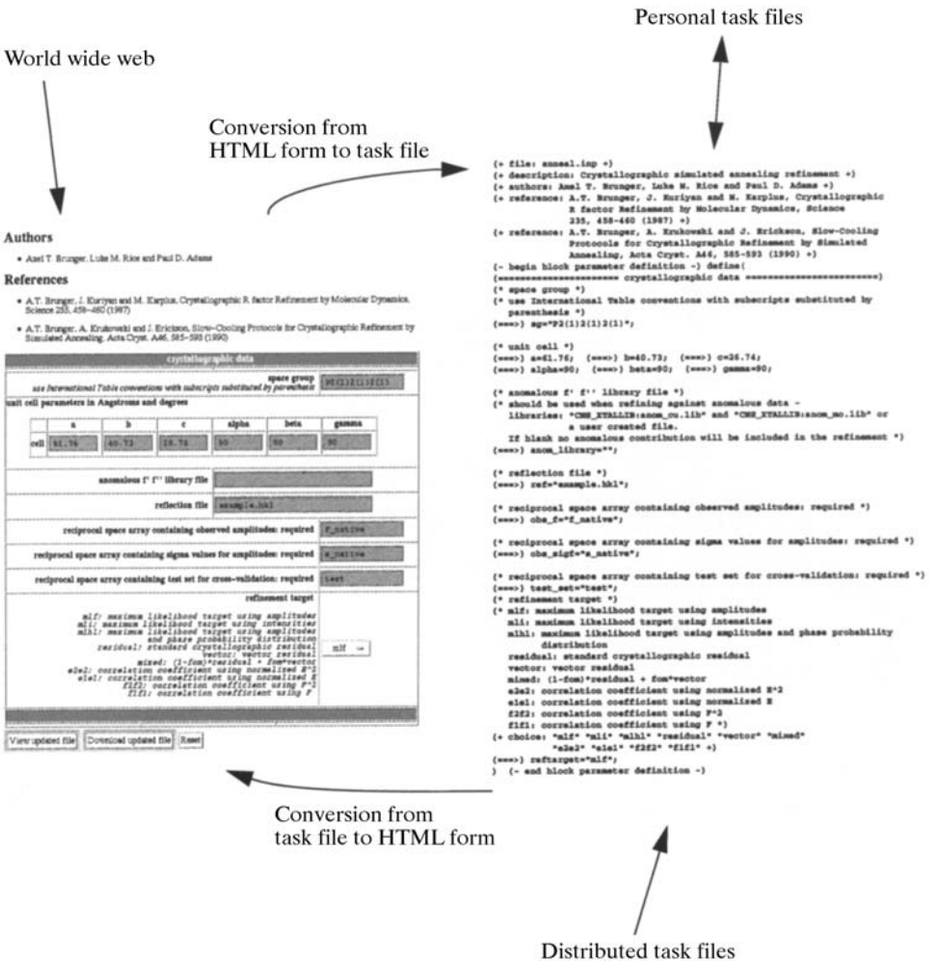

Fig. 25.2.3.10. Use of the CNS HTML form page interface, emphasizing the correspondence between input fields in the form page and parameters in the task file.

validated σ_A estimates, determines the weighting scheme between the X-ray refinement target function and the geometric energy function (Brünger *et al.*, 1989), refines a flat bulk solvent model (Jiang & Brünger, 1994) and an overall anisotropic B value for the model by least-squares minimization, and subsequently refines the atomic positions by simulated annealing. Options are available for specification of alternate conformations, multiple conformers (Burling & Brünger, 1994), noncrystallographic symmetry constraints and restraints (Weis *et al.*, 1990), and 'flat' solvent models (Jiang & Brünger, 1994). Available target functions include the maximum-likelihood functions MLF, MLI and MLHL (Pannu & Read, 1996*a*; Adams *et al.*, 1997; Pannu *et al.*, 1998). The user can choose between slow cooling (Brünger *et al.*, 1990) and constant-temperature simulated annealing, and the respective rate of cooling and length of the annealing scheme. For a review of simulated annealing in X-ray crystallography, see Brünger *et al.* (1997).

During simulated-annealing refinement, the model can be significantly improved. Therefore, it becomes important to recalculate the cross-validated σ_A error estimates (Kleywegt & Brunger, 1996; Read, 1997) and the weight between the X-ray diffraction target function and the geometric energy function in the course of the refinement (Adams *et al.*, 1997). This is important for the maximum-likelihood target functions that depend on the cross-validated σ_A error estimates. In the simulated-annealing task file, the recalculation of σ_A values and subsequently the weight for the crystallographic energy term are carried out after initial energy minimization, and also after molecular-dynamics simulated annealing.

25.2.3.10. *Conclusions*

CNS is a general system for structure determination by X-ray crystallography and solution NMR. It covers the whole spectrum of methods used to solve X-ray or solution NMR structures. The multi-layer architecture allows use of the system with different levels of expertise. The HTML interface allows the novice to perform standard tasks. The interface provides a convenient means of editing complicated task files, even for the expert (Fig. 25.2.3.10). This graphical interface makes it less likely that an important parameter will be overlooked when editing the file. In addition, the graphical interface can be used with any task file, not just the standard distributed ones. HTML-based documentation and graphical output is planned in the future.

Most operations within a crystallographic algorithm are defined through modules and task files. This allows for the development of new algorithms and for existing algorithms to be precisely defined and easily modified without the need for source-code modifications.

The hierarchical structure of CNS allows extensive testing at each level. For example, once the source code and CNS basic commands have been tested, testing of the modules and task files is performed. A test suite consisting of more than a hundred test cases is frequently evaluated during CNS development in order to detect and correct programming errors. Furthermore, this suite is run on several hardware platforms in order to detect any machine-specific errors. This testing scheme makes CNS highly reliable.

Algorithms can be readily understood by inspecting the modules or task files. This self-documenting feature of the modules provides a powerful teaching tool. Users can easily interpret an algorithm and compare it with published methods in the literature. To our knowledge, CNS is the only system that enables one to define symbolically any target function for a broad range of applications, from heavy-atom phasing or molecular-replacement searches to atomic resolution refinement.

25.2.4. The *TNT* refinement package

(D. E. Tronrud and L. F. Ten Eyck)

25.2.4.1. *Scope and function of the package*

TNT (Tronrud *et al.*, 1987) is a computer program package that optimizes the parameters of a molecular model given a set of observations and indicates the location of errors that it cannot correct. Its authors presume the principal set of observations to be the structure factors observed in a single-crystal diffraction experiment. To complement such a data set, which for most macromolecules has limitations, stereochemical restraints such as standard bond lengths and angles are also used as observations.

A molecule is parameterized as a set of atoms, each with a position in space, an isotropic B factor and an occupancy. The