The Declarative Expression of Semantic Integrity in a Database of Protein Structure

S. M. Embury and P. M. D. Gray

Department of Computing Science
King's College, University of Aberdeen
Aberdeen, Scotland, U.K. AB9 2UE
E-Mail: {sme|pgray}@csd.abdn.ac.uk
URL: http://www.csd.abdn.ac.uk/~pfdm

Abstract

This paper describes the use of a declarative constraint language to describe the semantics of a complex scientific domain, three-dimensional protein structure data, and assesses the requirements for integrity constraint mechanisms in this application area. The constraint language is based on the functional data manipulation language Daplex, and we use a range of examples from the protein database to illustrate its expressive power. We also consider the kinds of constraints that arise in protein structure data, thereby highlighting those which require particular attention when attempting to generate efficient checking code for them, and the requirements on the manipulation of the constraint information itself.

Keywords: integrity constraints, scientific databases, data semantics.

1 Introduction

Semantic integrity constraints can provide a useful extra source of information for both database management systems and applications, leading to more intelligent support for data manipulation. Many proposals for the use of such semantic domain information have appeared in the literature, ranging from semantic query optimisation [17], incorporating user preferences into queries [13] and declarative support for database updates [7] to safe transaction synthesis [22] and automatic transaction repair [12]. However, all these proposals depend upon the assumption that DBMS's will have a constraint language by which users can describe their semantic constraints in a declarative fashion, and an efficient method of ensuring that the data stored in the database satisfies all the constraints. Despite much research in this area, most database systems provide no support for declaratively-specified constraints, or can at best handle only a restricted set of the possible constraint forms. This fact, combined with the difficulty of eliciting semantic information from domain experts, has meant that there have been very few studies of the requirements of a constraint maintenance subsystem for real applications.

In this paper, we have tried to remedy this situation for one application area by examining the types of constraints that occur in a complex scientific domain, namely three-dimensional protein structure, and the kind of interaction with the constraint metadata that such a domain demands. We have based our study on the P/FDM database system [14], which is being used to store over 370Mb of protein structure data, and as the basis for an application which assists in the modelling of proteins of unknown structure [16]. Our constraint language is based on the functional-style data definition and manipulation language Daplex [23] for which a full compiler for the P/FDM system exists.

In the remainder of this paper, we give a brief introduction to the protein structure domain (Section 2) and describe the constraint language extension to Daplex (Section 3). We then use this language to give examples of the semantic constraints present in the protein domain (Section 4) and conclude by summarising the requirements for constraint maintenance that our study has highlighted (Section 5).

2 Problem Domain: Protein Structure

Proteins are an important class of organic macromolecules, that play a range of roles in the functioning of living systems. A protein consists of one or more chains of *amino acid residues*, which are folded in such a way that the interface that is formed by the surface of the molecule exhibits some useful biochemical property or behaviour. Each amino acid residue consists of two subunits. One of these is common to all residues, and it is this subunit which is covalently bonded to at most two other residues in order to form a continuous chain (called the *backbone* chain). The second subunit is bound to the backbone subunit at an atom called the α-carbon, and is called the *side chain*. There are only twenty different types of amino acid side chain that occur naturally in proteins, each being slightly different in size or having slightly different chemical properties. However, the number of potential sequences of these twenty amino acids is very large indeed (protein chains will typically contain anything from 100 to 1000 residues) and it is this flexibility that allows the same basic structure to have such a diverse range of functions and properties.

Protein structure is described in terms of four levels. The first level, called the *primary structure*, is given by the sequences of amino acid residues that make up the chains of a particular protein. It is known that the three-dimensional shape which a protein adopts is determined almost entirely by its amino acid sequence, therefore, in some sense, the primary structure gives a complete description of the structure of a particular protein.

A description of a protein at the second level (its *secondary structure*) gives a localised indication of the fold adopted by a particular segment of chain. There are two types of regular structure which recur in many different proteins because of their structural stability. One of these structures is the *helix*, which is stabilised by a set of non-covalent interactions called hydrogen bonds, occurring along its length. There are several varieties of helix, which differ only in the handedness (i.e. the direction) and the tightness of their coil. The most commonly occurring is the α-helix, which is illustrated in Figure 1 (a).

The second type of favoured structure is called the β-sheet. Sheets are formed by strands of chain lying roughly parallel to each other and stabilised by hydrogen bonds linking adjacent strands (see Figure 1 (b)). The secondary structure of a protein is given by stating which subsequences of the primary structure adopt helical conformations, which form strands within β-sheets, and which form the loops between these elements.

The third level of description, the *tertiary structure*, is the specification of the relative three-dimensional positioning of all the atoms within a protein. The final level, describing the *quaternary structure*, specifies the relative positions of the individual chains within a protein.

Each of these four levels gives a different perspective on a particular protein, and they are all therefore represented explicitly in the P/FDM protein database, the schema of which is illustrated diagrammatically in Appendix A.

Protein structure information has the characteristics of both design data [15] and scientific data [24], in that it is basically hierarchical in nature, it involves complex semantic integrity constraints and the interpretation of the data generally requires a combination of both data retrieval and computation. The storage of experimentally-determined data does not, in itself, require much support for complex updates, since the majority of changes to this data will be additions of new structures as they are published. The programs which analyse the data, however, do involve complicated updates, especially those like the model building program just mentioned, which are required to test complicated structural hypotheses against existing data. The domain of protein structure also involves a rich set of integrity constraints, ranging from the simplest domain constraints to complex biochemical and structural rules, which must be maintained if the analysis of the data is to produce useful information.

3 The Constraint Language Extension to Daplex

Our constraint language is an extension of the functional-style data definition and manipulation language Daplex. It is also based on the CoLan constraint language described in [1] and [10]. Function composition provides a natural way to represent chains of relationships forming sets or values to be constrained, and the functional paradigm allows a natural representation of expressions involving arithmetic, aggregate and set-based operators. Daplex already contains a rich set of constructs for describing predicates over sets of database objects, and we have been able to reuse all of these in our constraint language. This has not only saved us considerable work during the implementation, but it also means that our constraint language fits naturally with full Daplex and should feel comfortable to users who are already familiar with this language.

In our constraint language, constraint specifications consist of two parts: a quantification part and an (optional) predicate part. The new syntactic constructs which are required to extend Daplex in this way are given below (in BNF):

```
<constraint_dec>
  ::= constrain <quant_part> <to_have>
                <predicate> ;
   |  constrain <quant_part> <exists> ;

<quant_part>
  ::= <quantifier> <named_set_expr>
                <quant_rest>

<quant_rest>
  ::= so that <quant_part>
   |  <quant_part>
   |  []

<exists>
  ::= exists | to exists | []
```

Here, <quantifier>, <named_set_expr>, <to_have>, <predicate> and the other non-terminals are defined by the existing Daplex language (the full syntax of which is given in [14]):

```
<quantifier>
  ::= each | all | some | any | no
   |  exactly <const>
   |  at least <const>
   |  at most <const>

<named_set_expr>
  ::= <var> in <set> [such that <predicate>]
                [as <type>]

<predicate>
  ::= <bool_term>
   |  <predicate> or <bool_term>
```

(a) (b)

Figure 1: The two most commonly occurring elements of secondary structure in proteins (a) the α-helix, and (b) the β-sheet, with hydrogen bonds shown as dotted lines. (This diagram is taken from [18].)

```
<bool_term>
  ::= <bool_fac>
   |  <bool_term> and <bool_fac>

<bool_fac>
  ::= [not] <bool_prim>

<bool_prim>
  ::= <comparison>
   |  <quantified_expr>
   |  <set_membership>
   |  <subclass_membership>
   |  ( <predicate> )

<to_have>
  ::= has | have | to have
```

This has meant that we have been able to reuse many of the existing compiler components (e.g. the parser, optimiser and code generator) in the implementation of the constraint language compiler, which has greatly speeded up its development.

Six different quantifiers are supported by the constraint language: the three standard quantifiers

for all: denoted by the keywords each and all, as in "all students are older than 16"

```
constrain each s in student
to have age(s) > 16;
```

there exists: denoted by some and any, e.g. "some staff member is a lecturer"

```
constrain some s in staff
to have position(s) = ''lecturer'';
```

not exists: denoted by no, e.g. "no postgraduate student is younger than 20"

```
constrain no p in postgrad
to have age(p) < 20;
```

and three numerical (or cardinality) quantifiers [5]

exists at least: denoted by at least <n>, e.g. "there are at least 2 secretaries on the staff"

```
constrain at least 2 s in staff
to have position(s) = ''secretary'';
```

exists at most: denoted by at most <n>, e.g. "the Computing Science Department can support at most 100 undergraduate students"

```
constrain at most 100 u in undergrad
to have name(dept(u)) = ''CS'';
```

exists exactly: denoted by exactly <n>, e.g. "the university has exactly one principal"

```
constrain exactly 1 s in staff
to have position(s) = ''principal'';
```

The language as specified by the syntactic extension to Daplex given above allows the expression of a very rich set of constraints, as we shall demonstrate in the next section. Our current compiler for this language is able to generate fragments of Prolog code which will check the entire database for validity of a particular constraint, and is also able to simplify the code for checking against individual updates for constraints with a single quantifier involving only single-valued comparisons. Simplification [19] is an important technique in producing efficient procedural constraint checks from high-level declarative constraint specifications as it allows much redundant database retrieval to be removed from the code. We hope, therefore, to extend the scope of our current simplifying compiler to a wider range of constraints. One of the aims of the present study of constraints in the protein database is to identify the classes of constraints which occur most commonly and for which simplification is most important.

4 Semantic Integrity Constraints in Protein Structure Data

Having described our constraint language, we now show how it can be used to define integrity constraints on the protein database and thereby illustrate the requirements of a semantic integrity sub-system for a complex scientific domain. These constraints fall into three basic categories:

- constraints which reduce the allowable domain of an attribute from that specified by its type,
- constraints which guarantee the consistency of duplicated data, and
- constraints which describe biochemical "rules" about protein structure.

There are many constraints on the protein data, especially of the first two types, and we give only a representative selection here.

There are several attributes in the protein database schema that store numerical properties of the data; for example, the molecular_weight and resolution attributes of the class protein, and the molwt and accessible_area attributes of the class chain. All these attributes are defined as either integers or floats, whereas in fact they have much smaller domains, since none of them can take a negative value. We can use semantic constraints to reduce the domains of these attributes as follows:

```
constrain each p in protein
to have molecular_weight(p) > 0 and
              resolution(c) > 0;
```

```
constrain each c in chain
to have molwt(c) > 0 and
                accessible_area(c) > 0;
```

We can further constrain the molecular weight of proteins since we know that this value must be either equal to or greater than the sum of the molecular weights of its constituent chains:

```
constrain each p in protein
to have molecular_weight(p) ≥
    sum(molwt(component_protein_inv(p)
                    as chain));
```

We cannot say that the protein's molecular weight should always equal the sum of the weights of its chains, as it may contain other chemical groups (called ligands) for which the molecular weight is not recorded.

For some attributes we can specify an upper as well as a lower bound. The `pos/1` function defined on `residue` gives us an example of this kind of constraint. This function gives the position of a particular residue along a chain, with the first residue being at position 1, and the remainder being numbered consecutively according to their order of occurrence in the chain. The `num_residues(chain)` attribute stores a count of the residues within each chain, and therefore provides us with an upper bound for the `pos(residue)` function:

```
constrain each r in residue
to have pos(r) ≥ 1 and pos(r) ≤
    num_residues(has_component(r) as chain);
```

For extra security, we can make sure that the number of residues per chain is recorded accurately by the following constraint:

```
constrain each c in chain
so that num_residues(c) =
                count(has_component_inv(c));
```

Several of the relationship functions in the protein database represent symmetrical relationships, and we can enforce this essentially structural property using integrity constraints. The `parallel/1` function, for example, defined on the class `strand` relates particular strand instances to those of their neighbouring strands which run in a parallel direction to them. Similarly, `antiparallel/1` is a function relating neighbouring strands that are antiparallel to each other. The symmetry of both these relationships can be expressed as:

```
constrain each s in strand
to have s in parallel(parallel(s));
```

```
constrain each s in strand
to have s in antiparallel(antiparallel(s));
```

where `in` is the keyword which expresses set membership. We can use similar constraints to express the fact that no strand is both parallel and antiparallel to another strand[1]:

```
constrain no s in strand
to have s in (parallel(antiparallel(s))
        union antiparallel(parallel(s)));
```

and that every strand is either parallel or antiparallel to at least one other strand (i.e. strands cannot exist in isolation):

```
constrain each s1 in strand
to have some s2 in strand has
                parallel(s1) = s2 or
                antiparallel(s1) = s2;
```

The secondary structure hierarchy and its related functions offer scope for several interesting integrity constraints. Recall that at this level of description, each protein chain is divided into sections of a helical, strand or loop conformation. For example, we can state that the `first_structure` of a particular chain is the secondary structure element that is preceded by no other element in that chain:

```
constrain each s in structure such that
    no s2 in structure has follows(s2) = s
to have first_structure(structure_chain(s))
                            = s;
```

Similarly, we can express the fact that the last structure is that which has no succeeding element as the constraint[2]:

```
constrain each s in structure such that
    no s2 in structure has
                    follows_inv(s2) = s
to have end(s) =
        num_residues(structure_chain(s));
```

A further constraint on secondary structure is that there should be no gaps or overlaps in the assignment between the first structure and the last:

```
constrain each s in structure
to have end(s) = start(follows(s)) - 1 and
        start(s) = end(follows_inv(s)) + 1;
```

Notice here that we apply a weak-translation [25] to universally-quantified constraints in Daplex so that a constraint is considered to be trivially satisfied if any of the attributes on the quantified variable are undefined. Given a strong translation of the above constraints, under which constraints are always considered to be invalid if any of the attributes involved are undefined, we would have had to state explicitly that they are not applicable to the first and last structures, which will have a gap either before or afterwards:

[1] Notice that this formulation of the constraints relies on the knowledge that these relationships are both symmetrical.

[2] Since there is no explicit record of the last structure of a chain (i.e. no `last_structure` attribute) we identify it as the secondary structure element which ends with the last residue.

```
constrain each s in structure such that
    some s2 in structure has
                    follows_inv(s) = s2
to have start(s) = end(follows_inv(s)) + 1;
```

A weak translation on the other hand allows this kind of information to be specified implicitly, resulting in a more concise and natural constraint formulation.

The protein database contains several examples of constraints that maintain the consistency of duplicated data. This usually takes the form of a single function that abbreviates a relationship between two classes formed by a sequence of other relationships. We give a couple of examples here to illustrate the general form that this type of constraint takes. The first maintains the consistency of the `residue_structure(residue)` function, which materialises the relationship between residues and the element of secondary structure to which they have been assigned:

```
constrain each r in residue to have
    pos(r) >= start(residue_structure(r))
    and
    pos(r) =< end(residue_structure(r));
```

The second maintains consistency of the cycle of relationships linking β-sheets to the protein in which they occur:

```
constrain each s in sheet
to have sheet_protein(s) in
    component_protein(structure_chain(
                    strand_sheet_inv(s)));
```

The importance of the kinds of constraint illustrated here must not be under-estimated. Mistakes can and do find their way into any sizeable database — especially when, as in the case of scientific databases, the data passes through several hands and formats before it enters the database. An error recently located in the protein database, for example, caused several of the largest proteins stored to have apparent molecular weights of less than that of the smallest protein. The problem was caused by the leftmost digit of the molecular weight field being truncated by the program that prepared the data for input into P/FDM — an error that a simple constraint such as that given earlier on the molecular weight of proteins could have trapped immediately.

Finally, we consider the possibilities for using integrity constraints to express some of the *biochemical* semantics of the protein structure domain. While there are many potential constraints on the structure of proteins, not all of these are suitable for use as integrity constraints. It would not be practical, for example, to check all the possible constraints on the distances between bonded atoms, or the values of all the ϕ and ψ torsion angles along a chain, every time a new atom instance was created. Nor is it clear that such checks would be useful given that atoms are rarely manipulated individually. A better way to check this kind of low-level constraint is to use one of the special-purpose programs that have been developed for identifying potential anomalies in protein structure data. Procheck [18], for example, is a suite of programs that analyses the structure of an individual protein (input as a file in the standard Brookhaven file format [3]) and gives some estimate of its stereochemical quality. The programs also produce several graphical representations of the protein which can highlight "problem regions" within the structure. Programs like this can be used, therefore, to complement an integrity constraint mechanism by taking on the responsibility for checking the important atom- and residue-level constraints, which can involve a significant amount of computation and would have a serious effect on system performance if checked with integrity constraints. The DBMS is then left with the much simpler task of enforcing the kind of constraints specified above, and of any less complex or higher-level biochemical constraints.

The constraints that describe the semantics of disulphide bridges are good examples of the latter type of constraint. A disulphide bridge occurs when the sulphur atoms of two cysteine atoms approach each other closely enough for a covalent bond to form between them. Because of the strength of this bond (as opposed to the weaker, non-covalent interactions like hydrogen bonds) disulphide bridges play an important role in stabilising the conformation of protein chains. The semantics of this kind of interaction can be described by the following integrity constraints:

1. a disulphide bridge forms a symmetric relationship between two sub-components

   ```
   constrain each sc in sub_component
   to have disulphide(disulphide(sc))=sc;
   ```

2. the only type of natural residue that can participate in disulphide bridges is cysteine:

   ```
   constrain each r in residue such that
           some sc in sub_component has
                       disulphide(sc) = r
   to have name(r) = ''cys'';
   ```

   We cannot express this constraint over all sub-components, because there is a possibility that a synthetic group may be able to form a disulphide bridge with some other sub-component. (Synthetic groups are modelled as `sub_component` instances that are not also instances of `residue`.)

3. if two residues are joined by a disulphide bridge, their sulphur atoms must be close enough to be covalently bonded:

   ```
   constrain each r in residue
   to have distance(atom(r, ''sg''),
       atom(disulphide(r), ''sg'')) < 3.7;
   ```

where 3.7 is the sum of the van der Waals radii for two atoms of sulphur (given by Pauling [21] as 1.85Å). As in the case of the previous constraint, the scope of this constraint is limited by the possibility of synthetic groups being involved in the disulphide, whose sulphur atom may have some name other than "sg" (the standard name for the sulphur atom of cysteine).

All of these constraints are simple to check, with only the final constraint involving any computation (and, under constraint simplification, even this would occur only when the residue involved in the update was actually involved in a disulphide bridge.)

It is possible to give similar constraints for the other non-covalent interactions common in proteins (hydrogen bonds and salt bridges), although they are not so strictly constrained as the disulphide bonds and therefore result in more complex constraints. A salt bridge, for example, is an interaction between charged sub-components, and we could therefore specify the following constraints on natural residues:

```
constrain each sb in salt_bridge such that
            pos_source(sb) is a residue
to have name(pos_source(sb)) in
            {''lys'', ''his'', ''arg''};


constrain each sb in salt_bridge such that
            neg_source(sb) is a residue
to have name(neg_source(sb)) in
            {''asp'', ''glu''};
```

Unfortunately, there is an exception to this constraint in that the ends of protein chains also carry charge, and may therefore form chemically legitimate salt bridges that violate it. It is possible to avoid some of the problems caused by "exceptions" to constraints by taking care that the specification of the conditions under which the constraint is valid is accurate, as we did with the constraints on disulphide bridges. For instance, we could modify the above constraints on salt bridges so that they applied only when the source being constrained was not an end-residue.

Another way to deal with exceptions is to weaken the constraint in some way, as is shown by the following example involving hydrogen bonds in α-helices. The classic α-helix conformation is stabilised along its length by hydrogen bonds, linking each residue to the residue three places along the chain. We can express this information as a constraint on α-helices:

```
constrain each a in alpha
    each r in residue_structure_inv(a)
        such that pos(r) in
            {start(a) to end(a) - 3}
to have some h in hbond has
            acceptor(h) = r and
            pos(donor(h)) = pos(r) + 3 and
            acceptor_atom(h) = ''o'' and
            donor_atom(h) = ''n'';
```

The conformation usually adopted by helical sections of natural proteins, however, differs slightly from the classical α-helix conformation in a way that allows the carbonyl oxygen of each residue to also form hydrogen bonds with donors from outwith the helix [6]. In order to allow this type of helix to be stored in the database, we must modify our original, strong constraint to the following, weaker version:

```
constrain each a in alpha
    each h in hbond such that
        residue_structure(acceptor(h)
                    as residue) = a and
        residue_structure(donor(h)
                    as residue) = a and
        acceptor_atom(h) = ''o'' and
        donor_atom(h) = ''n''
to have pos(donor(h)) = pos(acceptor(h))+3;
```

Where our first constraint specified that all α-helices have a full set of hydrogen bonds stabilising their conformation, our second version requires only that the backbone hydrogen bonds which exist within the helix link the appropriate residues.

5 Conclusions

As can be seen from the above examples, the constraints that are extracted from a complex application domain are themselves complex and difficult to specify correctly. In such applications, we believe that flexibility of access to constraint metadata, and in particular the ability to delete and modify constraints from a working database, is vital if application developers are to be able to make confident use of an integrity constraint mechanism. The constraint-checking architecture which we have implemented for P/FDM [7, 10] gives us this flexibility.

Our study has also shown that the ability to modify the working set of constraints is also important in handling exceptions to constraints, which appear to play a rather different role in scientific databases than in more traditional application areas. In a scientific database, semantic integrity constraints can be seen as representing the scientist's current model of the part of the real world that is under investigation. Under this view, an exception to a constraint indicates some way in which the real world data behaves differently to what the scientist's model would predict. In other words, it points to a possible flaw in the model. Rather than provide some method of marking exceptional data, and allowing it to persist, as has been suggested for some traditional database systems [4, 2], we believe it is more important in scientific databases to allow constraints to be added and deleted freely at any time during the life of a particular database as the domain expert's model of the world evolves. In this way, the constraint checking mechanism is also helping to test the validity of the current model, since exceptions will be detected not only in the data that is initially present in the database, but also in any new experimental results that are added subsequently.

Our constraint language allows very complex constraints to be written with quantifiers nested to an arbitrary degree. Interestingly enough, the constraints we have encountered in the protein structure domain have not needed more than two nested quantifiers, and we are pleased to be able to cope with the complex integrity conditions that arise in such a scientific domain. We are developing a compiler for that language that is able to produce efficient (i.e. highly simplified) constraint checking code for constraints expressed in it. We have so far found our extension to Daplex to be able to express all the constraints we have encountered in the protein database, and our constraint checking architecture allows the possibility of handling this kind of simplified code. We now need to extend our simplifying compiler to be able to handle the main classes of constraint that our study has shown to be important for our application.

Another consequence of viewing the semantic integrity constraints as a representation of the scientist's current working model is that it must be possible to query the database metadata to discover what constraints are currently being enforced. Therefore, in P/FDM, we store both the declarative and procedural versions of each constraint in the metadata, and can display the declarative text of constraints to the user when violations are detected. Alternatively, the user can use Daplex itself to query the metadata [11], to discover for example all constraints on disulphide bridges, or all constraints involving the `follows` relationship.

Finally, a transaction mechanism, by which the checking of integrity constraints can be temporarily suspended while a sequence of updates are made, is essential if a database involving such complex constraints is to be practically useful. The initial ADAM implementation [14] did not have this capability, but in the later P/FDM implementation, we have been able to build a constraint maintenance subsystem with all these characteristics. Our current work now centres on the use of the semantic information expressed by constraints in a more generative way. In particular, we are exploring the use of constraint information in the optimisation of high-level declarative descriptions of complex database updates [8] and in suggesting repair strategies for long-duration design transactions [9].

Acknowledgements

The authors would like to thank Scott Leishman for his helpful comments on a draft of this paper, and Graham Kemp for his help in formulating the constraints given in Section 4.

References

[1] N. Bassiliades and P.M.D Gray. CoLan: a Functional Constraint Language and Its Implementation. *Data and Knowledge Engineering*, 1994. to appear.

[2] N. Bassiliades and I. Vlahavas. Modelling Constraints with Exceptions in Object-Oriented Databases. In J. Mylopoulos, editor, *Proceedings of 13th International Conference on the Entity-Realtionship Approach*, Manchester, December 1994.

[3] F.C. Bernstein, T.F. Koetzle, G.J.B. Williams, E.F. Mayer, M.D. Bruce, J.R. Rodgers, O. Kennard, T. Shimanouchi, and M. Tasumi. The Protein Data Bank: a Computer-Based Archival File for Macromolecular Structures. *J. Mol. Biol.*, 112:535–542, 1977.

[4] A. Borgida. Language Features for Flexible Handling of Exceptions in Information Systems. *ACM Transactions on Database Systems*, 10(4):565–603, December 1985.

[5] C.L. Chang. DEDUCE 2: Further Investigations of Deduction in Relational Data Bases. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 201–236. Plenum Press, 1978.

[6] T.E. Creighton. *Proteins: Structures and Molecular Properties.* W.H. Freeman and Company, second edition, 1993.

[7] S.M. Embury. *Constraint-Based Updates in a Functional Data Model Database.* PhD thesis, University of Aberdeen, King's College, Aberdeen, Scotland, July 1994.

[8] S.M. Embury and P.M.D. Gray. Non-Deterministic Updates in Daplex. Technical Report AUCS/TR9401, Dept. of Computing Science, University of Aberdeen, Aberdeen, Scotland, AB2 9UE, January 1994.

[9] S.M. Embury and P.M.D. Gray. Towards a Constraint-Based Repair Mechanism for Complex Database Updates. Technical Report AUCS/TR9404, Dept. of Computing Science, University of Aberdeen, Aberdeen, Scotland, AB2 9UE, July 1994.

[10] S.M. Embury, P.M.D. Gray, and N.D. Bassiliades. Constraint Maintenance Using Generated Methods in the P/FDM Object-Oriented Database. In Paton and Williams [20], pages 364–381.

[11] S.M. Embury, Z. Jiao, and P.M.D. Gray. Using Prolog to Provide Access to Metadata in an Object-Oriented Database. In C. Moss, editor, *International Conference on the Practical Application of Prolog*, London, April 1992. Applied Workstations, Ltd. 17 pages.

[12] P. Fraternali and S. Paraboschi. A Review of Repair Techniques for Integrity Maintenance. In Paton and Williams [20], pages 332–346.

[13] T. Gaasterland and J.Lobo. Qualified Answers that Reflect User Needs and Preferences. In J.Bocca, M. Jarke, and C. Zaniolo, editors, *VLDB 94 Conference*, pages 309–320, Santiago de Chile, September 1994. Morgan Kaufmann Publishers, Inc.

[14] P.M.D. Gray, K.G. Kulkarni, and N.W. Paton. *Object-Oriented Databases: a Semantic Data Model Approach.* Prentice Hall Series in Computer Science. Prentice Hall International Ltd., 1992.

[15] R.H. Katz. Computer-Aided Design Databases. In G. Ariav and J. Clifford, editors, *New Directions for Database Systems*, pages 110–123. Ablex Publishing Corp., 1986.

[16] G.J.L. Kemp. Protein Modelling: a Design Application of an Object-Oriented Database. In J. Gero, editor, *Proc. 1st Int. Conf. on Artificial Intelligence in Design*, pages 387–406. Butterworth-Heinemann, 1991.

[17] J.J. King. *Query Optimisation by Semantic Reasoning.* UMI Research Press, 1984.

[18] R. Laskowski, M.W. MacArthur, D.S. Moss, and J.M. Thornton. PROCHECK: a Program to Check the Stereochemical Quality of Protein Structures. *J. Appl. Cryst.*, 26:283–291, 1993.

[19] J.-M. Nicolas. Logic for Improving Integrity Checking in Relational Databases. *Acta Informatica*, 18:227–253, 1982.

[20] N.W. Paton and M.H. Williams, editors. *Proc. of 1st Int. Workshop on Rules in Database Systems (RIDS '93)*, Workshops in Computing, Edinburgh, August 1993. Springer-Verlag.

[21] L. Pauling. *The Nature of the Chemical Bond.* Cornell University Press, second edition, 1949.

[22] X. Qian. The Deductive Synthesis of Database Transactions. *ACM Transactions on Database Systems*, 18(4):626–677, December 1993.

[23] D.W. Shipman. The Functional Data Model and the Data Language DAPLEX. *ACM Transactions on Database Systems*, 6(1):140–173, March 1981.

[24] A. Shoshani, F. Olken, and H.K.T. Wong. Characteristics of Scientific Databases. In U. Dayal, G. Schlageter, and L. H. Seng, editors, *VLDB 84 Conference*, pages 147–160, Singapore, 1984.

[25] S.D. Urban. ALICE: an Assertion Language for Integrity Constraint Expression. In *Proceedings of Conference on Computer Software Applications*, September 1989.

A The Protein Database Schema