

Techniques for effective integration, maintenance and evolution of species databases

Andrew C. Jones, Iain Sutherland,
Suzanne M. Embury and W. Alex Gray
Department of Computer Science
Cardiff University
PO BOX 916
Cardiff, CF24 3XF, UK
{Andrew.C.Jones|I.Sutherland}@cs.cf.ac.uk
{S.M.Embury|W.A.Gray}@cs.cf.ac.uk

Richard J. White and John S. Robinson
Biodiversity & Ecology Research Division
School of Biological Sciences
University of Southampton
Southampton, SO16 7PX, UK
{R.J.White|J.S.Robinson}@soton.ac.uk

Frank A. Bisby and Sue M. Brandt
Biodiversity Informatics Laboratory
Centre for Plant Diversity & Systematics
The University of Reading
Reading RG6 6AS, UK
{F.A.Bisby|S.M.Brandt}@reading.ac.uk

Abstract

The LITCHI (Logic-based Integration of Taxonomic Conflicts in Heterogeneous Information systems) project is concerned with the integration and maintenance of databases of biological knowledge organised by species. We have identified a number of constraints pertaining to good taxonomic practice that can be applied to individual species databases in order to determine taxonomic conflicts. These constraints can also be applied to a database formed by merging species databases from distinct sources. As the project has progressed, we have identified a number of distinctive features of the problem domain and needs of the intended users which have had a significant impact on the techniques and modes of operation that we found to be appropriate, especially in contrast with applications that handle rapidly-accumulating 'raw' data. It is upon these aspects of LITCHI that we concentrate in the present paper, viewing LITCHI as an example of the more general problem of merging scientific data sets in which there are conflicts between the terminology used in the various data sets.

Taxonomic knowledge is characterised by being slowly added to and modified, and by being subject to disagreement among experts. For the knowledge to be useful, it is essential that a biologist should be able to work with full knowledge of such conflicts of opinion. In LITCHI we

provide a tool which may be used by a taxonomic editor, over a potentially long transaction period of perhaps a few months, in order to identify taxonomic conflicts between species databases and resolve them in an appropriate way. In this paper we explain how LITCHI has been designed in order to detect and allow the user to resolve these taxonomic conflicts.

1. Introduction

Taxonomy may be defined as 'the study and description of the variation of organisms, the investigation of the causes and consequences of this variation and the manipulation of the data obtained to produce a system of classification' [13]. In this paper we present the LITCHI¹ system. LITCHI is a tool we have developed with the aim of helping taxonomists to test checklists of scientific names for conflicts and hence (i) to improve the data quality in the taxonomic databases from which the checklists were obtained, and (ii) to provide a basis for integration of taxonomic databases. We provide a survey of the techniques we have had to employ and develop in order to build a tool that achieves these aims.

Taxonomy is just one of many scientific areas in which

¹Logic-based Integration of Taxonomic Conflicts in Heterogeneous Information systems

consistent nomenclature is important: other examples include planetary nomenclature [7], geographical nomenclature (e.g. [12]) and gene nomenclature (e.g. the SGD Gene Name Registry²). In all these domains it is important to be able to refer to the entities of interest unambiguously. We exploit in our system the fact that it has been possible to develop constraints that detect cases in which conflicts occur by inspection of the scientific names used and the relationships between them. In LITCHI we are not concerned with interpretation of vast quantities of fast-accumulating data using OLAP [4] techniques, for example; rather, we are providing a tool for maintaining a consistent classification scheme in the face of conflicting and changing expert opinion. It is only with such a scheme that biological data can be associated with the appropriate species.

In this paper we shall commence by explaining in more detail the nature of the scientific problem that LITCHI is addressing and the challenges that have to be met. We shall then outline how we have approached this problem in LITCHI and compare our approach, in general terms, with other approaches that have been taken in the general area of consistency of taxonomic databases. LITCHI solves problems both of constraint violation detection and of repair. We explain how constraint violation detection is improved by the use of constraints that may have exceptions and of orthographic testing. In the repair process, LITCHI is designed to support long intermittent transactions, to keep the number of repair actions small by typing of constraints, and to allow the user a flexible approach to the repair of these constraints. In the concluding section we discuss the relevance of our techniques to other domains and identify areas of possible future research.

2. The scientific problem

The discipline of taxonomy benefits from an agreed general scheme for classification (the taxonomic hierarchy) and agreed schemes for naming the *taxa* (singular: *taxon*) thus produced, such as the codes of botanical [6], zoological [11] and bacterial [10] nomenclature. The taxonomic hierarchy has various *ranks*, such as (in increasing order of specialisation) *family*, *genus*, *species* and *subspecies*. One of the tasks of the taxonomist is to determine what range of variation is exhibited by the individuals (e.g. plants or animals) that are included in a taxon at each of the hierarchical ranks. Having done this, the appropriate code of nomenclature imposes constraints on the nature of the scientific names (s)he may choose for the taxa, and their relationships to each other. A *checklist* comprising the names assigned can then be produced, which includes for each taxon the *accepted name* and any synonyms that others have used. Another

taxonomist may produce a conflicting checklist for a number of reasons, as illustrated in figure 1. These checklists are lists of species. A species name has two parts, namely its *genus* and its *epithet*, and has an associated authority. For example, *Vicia faba* L. has genus *Vicia*, epithet *faba* and authority 'L.' (which is an agreed abbreviation for Linnaeus). In each case, the *accepted* name is listed, followed by an indented list of *synonyms* that pertain to the same taxon. One conflict in this case is that *Vicia faba* L. is regarded as an accepted name in one list, but as the synonym of another accepted name in the other list. Similarly, of course, there is another conflict: *Faba bona* Medikus is regarded as an accepted name in one list, but as the synonym of another accepted name in the other list.

List 1:

```
...
Vicia faba L.
    Faba bona Medikus
    Faba faba (L.) House
    Faba major Desf.
    Faba minor Roxb.
    Faba sativa Bernh.
    Vicia vulgaris Gray
```

...

List 2:

```
...
Faba bona Medikus
    Faba faba (L.) House
    Orobus faba Brot.
    Vicia esculenta Salisb.
    Vicia faba L.
    Vicia vulgaris Gray
```

...

Figure 1. Conflicting extracts from two checklists

Taxonomic databases are an important resource for biologists. Many of these databases are *species databases*, i.e. biological data about organisms arranged by species, with little information about other taxonomic ranks. With the advent of taxonomic databases there is the attractive prospect of merging them to build a global species information system [2]. But the availability of taxonomic databases both intensifies, and yet provides the basis of a potential solution to, a fundamental problem: how is it possible to gather data associated with a given species from various sources unless (i) we know that a given species name does in fact refer to the same species in each source and (ii) we are aware

²http://genome-www.stanford.edu/Saccharomyces/gene_guidelines.html

of any other names by which the species is known in these sources? The data is far more readily available than when it was only held in printed form – in other words, we are now more likely to *uncover* taxonomic problems than previously! But taxonomic databases provide the basis for a potential solution to this problem in that the large quantities of information available can be processed electronically in order to identify conflicts – if suitable techniques can be developed. It is the development of such techniques that has been central to the LITCHI project.

The *merging of checklists* is an important step towards the *integration* of taxonomic databases. Making a consistent checklist available containing every name in each database of interest means that the user will have available an *index* which may be used to locate data for a given species in each database.³ Figure 2 shows one possible way of merging the two checklist extracts given in figure 1. In the taxonomist's judgement, *Vicia faba* L. is the accepted name in this case, but note that we have not *deleted* any names. Thus we know that if we wish to find what is known about *Vicia faba* L. then we also need to query taxonomic databases using *Faba bona* Medikus, *Faba faba* (L.) House, etc. Moreover, a scientist knowing the species by the name *Orobus faba* Brot., for example, can discover from the checklist that it is referred to by the name *Vicia faba* L. in the taxonomic treatment that the combined checklist represents.

...

Vicia faba L.

Faba bona Medikus
 Faba faba (L.) House
 Faba major Desf.
 Faba minor Roxb.
 Faba sativa Bernh.
 Faba vulgaris Moench
 Orobus faba Brot.
 Vicia esculenta Salisb.
 Vicia vulgaris Gray

...

Figure 2. A possible way of merging the checklist fragments in figure 1

As illustrated above, it is both possible and desirable to merge checklists to form a consistent taxonomic view. Because there is a notion of good taxonomic practice that is

³The *merging* of such data, resolving conflicts at the data level, is a further difficulty that is outside the scope of the present project, but we have already made a significant step forward in that the biologist is given the means of determining *which* data pertain to the species of interest, using human judgement to compare the data thus located.

manifest in the names used and their relationships, there is the opportunity to make these notions explicit in a formal model of taxonomic practice – we have chosen to express these notions as constraints. But there are a number of problems associated with this that arise out of characteristics of taxonomy:

1. Taxonomists may differ considerably over certain taxonomic groups. Thus there may be many conflicts between large checklists that are very hard to detect by hand.
2. Taxonomy is a discipline subject to change. For example, the name applied to a given specimen may change due to discovery of publications of which the taxonomist was previously unaware describing the taxon to which it belongs, or due to change of taxonomic opinion. The problem with this is that if the accepted name changes, it will no longer be possible to access data (or literature) associated with the previous name unless it is associated with the current one. This is why in merging checklists no names are deleted (unless they are simply *errors*).
3. There are special cases to some general rules of taxonomy. This poses the problem of how to handle these in a constraint-based system without making the constraints unwieldy in complexity.
4. There are variations in spelling of scientific names and authorities. Moreover, there are numerous possibilities for how authorities may be abbreviated, how an authority comprising more than one person is signified, etc. This poses an additional problem for the constraint checker when seeking to determine whether two names are the same or different.
5. Theoretically there may be many ways of repairing a given conflict, most of which will be invalid taxonomically. This means that it is important to be able to narrow down the set of possible repairs presented to a user as much as possible, in order to avoid confusion.
6. A taxonomist may use an arbitrary amount of contextual information and scientific knowledge to help him or her solve a given taxonomic conflict. This means that simply presenting the user with the data that participates in a given conflict may not be sufficient.
7. A taxonomic editor will not be an expert on every taxon in a given list, and will need to consult other experts over a period of time in order to determine the appropriate action to take. This poses the problem that one cannot assume the repair process will be completed in a single session – persistence is required.

We have thus identified a number of problems that need to be solved if the LITCHI software is to be useful to taxonomists. In the next section we shall outline the general approach taken in the implementation of LITCHI; in the subsequent sections we shall return to the problems listed in order to explain how we have solved them.

3. The LITCHI approach

The LITCHI software has been built with the aim of supporting the process of detecting and repairing inconsistencies in taxonomic checklists, including composite checklists formed by merging other checklists. We make use of constraint technology to perform this task. These constraints are expressed as Prolog rules, and have been elicited from taxonomic experts: they are constraints which a taxonomist would expect to see satisfied in a taxonomically consistent checklist. Details of this, and of how we determine what kinds of repairs can be carried out when a constraint is violated, are to be found in sections 4 and 5. In the present section we shall first explain how our approach is to be distinguished from a number of other approaches to dealing with problems of conflicts in taxonomy, and then outline a number of scenarios in which LITCHI can be used. These usage scenarios provide the context within which the techniques described in subsequent sections have had to be developed.

3.1. Related work

This is not the first time that the problems of consistency of taxonomic data have been investigated. But as far as we know, LITCHI is the first system to employ constraints to test for taxonomic consistency by analysing check-lists generated by species databases, and to help the user to repair conflicts by applying the same criteria as (s)he would have applied manually when merging checklists, i.e. creating a consistent checklist in which all the names that have been rejected as accepted names will appear as synonyms somewhere in the list. Indeed, as far as we are aware, this is the first application of constraint violation repair techniques to a ‘real-world’ problem.

Particularly relevant related work in the area of taxonomic databases is that of ReTAX [1], PROMETHEUS⁴, HICLAS [8] and the nomenclatural database integration system described by Kitakami *et al* [9]. In ReTAX, the aim is to test for consistency of specimen feature data with the taxa to which they are attributed. The emphasis is on the application of theory revision and conceptual clustering techniques to help resolve the inconsistencies found. In terms of inconsistency of terminology, these are detected

⁴<http://www.dcs.napier.ac.uk/~prometheus>

primarily because of inconsistencies between the data describing the members of each taxon contained in the system. The PROMETHEUS project takes the view that in essence a taxon is defined by its specimens, and proceeds to look at ways of grouping specimens, etc. The rules of nomenclature are implemented so that, for example, the correct name of a taxon can be inferred from nomenclatural details of the specimens it contains. In HICLAS, relationships between taxa are represented by nodes and operators that capture the kinds of operations that may be performed on a taxon – for example, splitting a species into two new species. In Kitakami *et al*'s system, the emphasis is upon building a consistent *hierarchy*.

In contrast, we are concentrating primarily upon the *species* level of the taxonomic hierarchy, and are looking solely at details of the names and their relationships in checklists generated from species databases. (Because of the structure of scientific names, in which the genus is an explicit component of a species name, implicit fragments of a taxonomic hierarchy are present). But our emphasis is upon determining what the leaf nodes of the taxonomic hierarchy should *be*, rather than attempting to build a full taxonomic hierarchy. It is important to reiterate that this approach is capable of yielding very useful checklists that can be used as indices into all species checklists that contributed to the combined, consistent list, and to the databases from which they were derived. This is because names are associated by taxon in the resultant checklist, i.e. they refer either to the same species or possibly (in the case of a synonym) to a taxon that covers a smaller range of variation than that to which the accepted name refers. No attempt is currently made in LITCHI to make this particular detail of the nature of the relationship between synonyms and an accepted name, or the relationships between synonyms of the same accepted name, explicit. We recognise that a permanent record of such detail could be helpful to the taxonomist in future versions of the system.

3.2. Litchi as a tool for integration, maintenance and evolution of taxonomic databases

LITCHI can be used both for testing individual checklists for consistency and for merging checklists then testing the resultant checklist for consistency. In both cases, LITCHI provides the facility to resolve any conflict detected. A typical scenario is as follows:

1. The user imports one or more taxonomic checklists, in a format such as XDF [15, 16], into the LITCHI checklist database.
2. The user instructs the LITCHI system to combine these checklists into a single checklist, ready for testing by the LITCHI software. Information is retained in the

combined checklist of the original list from which each taxon entry was obtained.

3. A Conflict Reasoning Engine (CRE) is invoked in order to identify taxonomic conflicts in this combined checklist. The CRE also determines a set of taxonomically valid updates associated with each conflict.
4. The user examines the conflicts, and builds up a set of updates that he or she wishes to make from the sets of possible updates generated by the CRE, by repeatedly selecting a particular conflict and answering a sequence of questions presented by LITCHI, the answers to which will determine the precise update to be performed.
5. The updates are performed and the revised checklist is checked by the CRE.
6. The user may well need to repeat steps 3 to 6 a number of times. The reasons for this are (i) the user may have introduced new inconsistencies into the checklist, and (ii) the user may not have been able to establish the appropriate actions to take in regard to some of the conflicts – perhaps, for example, a letter has to be written to a specialist in order to obtain his or her expert opinion in regard to the most appropriate resolution of a particular conflict.
7. When it has been decided that no further work can be done to resolve conflicts in the time-scale available, the user exports the combined checklist into a form such as XDF.

The above scenario is checklist-orientated, but can be used in a number of contexts:

checking, maintenance & evolution of taxonomic databases:

An individual taxonomic database may be in an inconsistent state and in need of checking by LITCHI. But also, in the process of time an individual taxonomic database will be subject to revision as new knowledge becomes available. It is therefore possible for inconsistencies to creep in – indeed, due to the nature of taxonomy, inconsistencies could be introduced into the taxonomy relating to the species already known to the system because, for example, their names may be changed. It is therefore desirable to apply LITCHI periodically to the checklists generated from such databases, and use the corrected checklists as the basis for reorganisation of the database when necessary.

integration of taxonomic databases: It is often desirable to merge taxonomic databases that have overlapping taxonomic coverage in order to create systems with greater taxonomic coverage. In this case, there may

be conflicts between the databases in the areas of overlap. By application of LITCHI to a merged checklist generated from the contributing databases, a consistent checklist can be generated which contains all the names in the contributing checklists, correctly associated with each other. This means that the checklist can then be used as an index into the taxonomic databases. We assume in the LITCHI project that the user of such an index will use his or her judgement in interpreting the data retrieved from each database, because there may be differences of representation and data-level conflicts that are outside the scope of the present project.

improvement of taxonomic database quality: One scenario of usage that we find attractive is that by using LITCHI it is possible to improve the quality of a checklist by comparing it with other checklists having similar taxonomic coverage. An example of why this is possible is that the additional checklist may contain names in a single taxon entry that associate names in two distinct taxon entries in the main list.

We have thus shown how LITCHI can be used. In the above scenarios it may be seen how LITCHI deals with the problem, mentioned in section 2, that taxonomy is a discipline subject to change. There are a number of challenges that we have had to meet in order to make a system such as LITCHI feasible, as we shall now see.

4. Checking for conflicts

In the LITCHI project we have been faced with the challenge of implementing a tool that helps taxonomic editors by detecting conflicts and allowing them to perform suitable repairs to resolve these conflicts. In this section we shall deal with three aspects of conflict detection that we have had to address. The first of these is elicitation of knowledge about taxonomic practice from taxonomic experts, and its expression as ‘hard’ constraints. We have gone into the detail of some of these constraints in previous papers [14, 5], so we only provide a brief example of the kind of constraint that has been elicited in order to illustrate the technique as one of several that have been used in LITCHI. The main point of this first section is that it has been *possible* to elicit constraints on taxonomic practice and that, without this, our whole approach to the LITCHI project would have been infeasible. We also overcome a problem listed in section 2, that detecting conflicts in large checklists by hand is very difficult. In the remaining two sections we present techniques we have not mentioned in previous publications. Despite the usefulness of ‘hard’ constraints, we have encountered examples of taxonomic practice in which there are ‘exceptions to the rule’, as indicated in section 2.

We shall illustrate how we accommodate such exceptions in our constraints where necessary. Another problem mentioned earlier is that, in checking for whether names are the same, there are some variations of spelling, etc., that do not necessarily mean we should regard the names as being distinct. We need to be able to perform constraint checking in the face of such complications, and we outline how we deal with this problem in the last sub-section below.

4.1. Elicitation of ‘hard’ constraints

Many of the constraints on taxonomically consistent checklists can be expressed as traditional ‘hard’ integrity constraints. The approach we have taken is to develop English versions of the constraints, express these in first-order logic and then translate them into Prolog rules which will be satisfied only if the constraints are violated. The Prolog rules effectively perform a query over the checklist database – using ProData as the link between Prolog and the database. In some cases the Prolog prototype rules have been replaced by SQL queries to obtain acceptable performance. Although these arrangements would be completely unsuited to real-time systems with very frequent updates, because all queries are over the entire database state, they work well for our purposes.

As an example, consider the conflict illustrated in section 2. The constraint that has been violated is:

A full name which is not a pro parte name⁵ may not appear as both an accepted name and a synonym in the same checklist.

This is expressed in first-order logic thus:

$$\begin{aligned}
 &(\forall n, a, l, c_1, c_2, t_1, t_2)(\text{accepted_name}(n, a, c_1, l, t_1) \\
 &\quad \wedge \text{synonym}(n, a, c_2, l, t_2) \\
 &\quad \Rightarrow \text{pro_parte}(c_1) \wedge \text{pro_parte}(c_2))
 \end{aligned}$$

or, in Prolog, as a rule which succeeds if the constraint is violated:

```
violation :-
  accepted_name(N, A, C1, L, T1),
  %Taxon T1 has acc. name N
  synonym(N, A, C2, L, T2),
  (\+ pro_parte(C1); \+ pro_parte(C2)).
```

As might be expected, elicitation of such constraints from taxonomists is a laborious, iterative process. Considering the above constraint, for example, our first version of the constraint was simply:

⁵Pro-parte names arise when it is decided that a taxon must be split into smaller taxa at the same taxonomic rank

A full name may not appear as both an accepted name and a synonym in the same checklist.

The special case of pro-parte names was elicited through further discussion. Although some constraints can be derived directly from the codes of nomenclature (so far we have restricted ourselves to the botanical code), this is not true in every case, particularly in relation to names that have been changed for scientific reasons. But a good number of such constraints *have* now been captured (33 at present), and have proved their worth in detecting conflicts – even in individual databases such as ILDIS⁶, which were previously believed to have internally-consistent taxonomies.

4.2. Constraints with exceptions

Although it is possible to express many of the constraints on good taxonomic practice as ‘hard’ constraints, as described above, this is not always the case. For example, according to the International Code of Botanical Nomenclature, all family names should have the ending *-aceae*, e.g. *Rosaceae*. But there are eight exceptions to this, an example of which is *Palmae* (instead of *Palmaceae*).

For another example, consider the following scientific names taken from the ILDIS checklist:

```
Vicia truncata Torrey & A. Gray
Vicia america subsp. americana Willd.
Vicia america subsp. truncata (Torrey & A. Gray) Love & Love
```

To understand this example it is necessary to know that a subspecies name comprises three parts – its genus, its specific epithet (which species it belongs to) and its subspecific epithet (which subspecies it is) – in addition to the authority associated with the subspecies and (possibly) the authority associated with the species to which it belongs. Note that the specific epithet of the first name listed (*truncata*) is the same as the subspecific epithet of the third name listed. According to the botanical code of nomenclature, in naming new taxa this practice should be avoided, but the code admits existing names of this kind. It is only in cases where the specific and subspecific epithets are the *same* (as in the second name, both epithets being *americana*) that this should occur.

We wish to enable users to indicate, when exceptions of the kinds indicated above occur, either that a genuine exception has been encountered or that it is, in fact, a manifestation of a taxonomic conflict. If it is a genuine exception, then we do not wish the CRE to report it to the user on future occasions. So the exception must be recorded somehow. To change the form of a constraint so that it does not classify

⁶<http://www.ildis.org>

any exceptional cases as violations is impracticable. This is because:

1. Every exception will require the inclusion of one or more terms within the rule to account for that exception.
2. Although, in the first example given above, there is a well-defined set of exceptions that is known at the start, an exhaustive list of exceptions of the sort given in the second example cannot be created and so the set of known exceptions must grow as LITCHI is used. This would mean updating the constraint on a regular basis.

The alternative is to record exceptions identified by the user separately. To illustrate our approach, consider the statement of the above constraint without reference to exceptional cases:

No subspecies of a given species should have the same subspecific epithet as the specific epithet of some *other* species in the same genus.

The first-order logic for this constraint is:

$$\neg(\exists g, s_1, s_2, l, a_1, a_2, c_1, c_2, t_1, t_2, r) \\ (name([g, s_1, subsp, s_2], a_1, c_1, l, t_1) \\ \wedge name([g, s_2|r], a_2, c_2, l, t_2) \wedge s_1 \neq s_2)$$

(For convenience, we have adopted a Prolog-like list notation to allow us to extract the sub-parts of a Latin name in first-order logic.) Our approach is to replace such a constraint ϕ , say, by *two* constraints: one which indicates that either the original constraint is satisfied or an exception should have been noted; and another which indicates that if an exception has been noted for particular values of the constraint variables then the original constraint should *not* be satisfied, i.e:

$$(\forall g, s_1, s_2, l, a_1, a_2, c_1, c_2, t_1, t_2, r) (\\ (\neg name([g, s_1, subsp, s_2], a_1, c_1, l, t_1) \\ \vee \neg name([g, s_2|r], a_2, c_2, l, t_2) \\ \vee s_1 = s_2 \\ \vee ex_\phi(g, s_1, s_2, l, a_1, a_2, c_1, c_2, t_1, t_2, r))$$

and:

$$(\forall g, s_1, s_2, l, a_1, a_2, c_1, c_2, t_1, t_2, r) (\\ (\neg ex_\phi(g, s_1, s_2, l, a_1, a_2, c_1, c_2, t_1, t_2, r) \\ \vee name([g, s_1, subsp, s_2], a_1, c_1, l, t_1)) \wedge$$

$$(\neg ex_\phi(g, s_1, s_2, l, a_1, a_2, c_1, c_2, t_1, t_2, r) \\ \vee name([g, s_2|r], a_2, c_2, l, t_2)) \wedge \\ (\neg ex_\phi(g, s_1, s_2, l, a_1, a_2, c_1, c_2, t_1, t_2, r) \\ \vee s_1 \neq s_2))$$

This can then be expressed in Prolog in the usual way – in this case, as clauses for a rule we have labelled as number 24. A typical clause, stating that the conflict between *Vicia truncata* Torrey & A. Gray and *Vicia americana* subsp. *truncata* (Torrey & A. Gray) Love & Love has been labelled as an exception to the rule would take the form:

```
ex_24('Vicia', america,
      truncata, <checklist-no>,
      '(Torrey & A. Gray) Love & Love',
      'Torrey & A. Gray',
      [], [], <taxon-id 1>, <taxon-id 2>,
      []).
```

The theoretical justification of this approach, and details of how we deal with the general case in which both universal and existential quantifiers are present, is outside the scope of this paper because our intention here is to give an outline of each of the major techniques used. But the main point is that in taxonomy, as in many real-world scenarios, some constraints cannot sensibly be expressed as constraints that must always be satisfied. We have a mechanism, illustrated above, that enables us to accommodate such special cases.

4.3. Dealing with orthographic problems

Many of the constraints in the LITCHI model of biological nomenclature require the comparison of the different parts of scientific names, in order to determine whether they are the same or different. Unfortunately, since we are modelling a system that was created by humans for use by humans, we cannot rely on a simple check for lexical equality in order to determine this. Checking for equivalence of the Latin parts of a name is easier than detecting equivalence of authorities, and yet there are a couple of complications to be taken into account:

- The letters ‘i’ and ‘j’ are considered to be equal, as are the letters ‘u’ and ‘v’, so that the names *Geastrum hygrometricum* and *Geastrvm hygrometrjcvum* should be recognised as the same name.
- The gender of the specific epithet should reflect the gender of the genus name. This means that the epithets *ambiguum* and *ambigua* should be recognised as being equivalent. An example of where this equivalence is important to detect is in testing for the relationship between the names *Trifolium ambiguum* and *Amoria ambigua*. *Amoria ambigua* is a synonym of *Trifolium*

ambiguum which arose because taxonomic opinion regarding the genus of this species has changed. It is for this reason that the epithets are the same, if the endings are ignored. If these two names appear in distinct taxa, then the system should be able to identify that the epithets are the same, although they have different endings, so that a conflict can be reported.

The approach we take to dealing with such variations is to implement a test for equality of scientific name components that ignores differences in ending or spelling of this sort. Since in some cases names that are genuinely distinct may be regarded as being the same, it follows that occasionally conflicts may be reported that are not true conflicts, but the taxonomist can exercise his or her judgement in dealing with these.

Equality of authority is rather more difficult to detect than equality of Latin names due to lack of agreement on the proper way to spell some authors' names and the propensity of different groups of taxonomists to abbreviate names in different ways. For example, consider the following pairs of author names, all of which refer to the same person:

William Jones	Jones
Keith Jones	K.Jones
Liou Liang	Lian Liu
Jwan Jakovlewitsch Akinfiev	Ivan Yakovlevic Akinfiyev

Clearly it is not practical to try to model all of the different conventions for names from the range of different countries that are required. Fortunately, however, taxonomists have already taken steps to try to standardise the use of author names in taxon names, and several lists of 'agreed' or 'recommended' abbreviations for author names have been collated (e.g. Brummit & Powell [3]). In the LITCHI project, we have stored two of these name lists within our repository of taxonomic knowledge in order to assist in comparing them, and we attempt to match author names against these lists of known and recommended abbreviations on import. This reduces the amount of matching that has to occur when the conflict detection process is being performed.

5. Repairing checklists

In the field of taxonomy, it is important to be aware of conflicts between taxonomic treatments, but it is also important to be able to resolve these conflicts and build a consistent taxonomic treatment that can be used by scientists. Thus it is desirable not only to *detect* conflicts, as described in the previous section, but to repair the checklists by *resolving* the conflicts. In order to be able to do this effectively, the LITCHI system must be capable of supporting long, intermittent transactions because one cannot assume

that the repair process will be completed in a single session (see section 2): a *repository* of checklist and other information is used for this purpose. In section 2 we also saw that when resolving conflicts it is desirable for the user not to be overwhelmed by repairs that, though theoretically possible, would not be taxonomically valid: we have developed a *conflict typing* technique to address this difficulty. In the last subsection we discuss how the LITCHI environment has been designed so that the user can have effective control over the conflict repair process, and so that (s)he can gain access to information which did not directly give rise to a conflict but may have some bearing on the repair action the user may conclude to be appropriate. Having this level of control and information solves the last two problems mentioned in section 2.

5.1. Need for repository

We mentioned some aspects of the LITCHI repository in a previous paper [5], but we have extended the repository substantially since then. A central database has been chosen as the means to communicate information between the various software components for import/export, detection of conflicts and the user interface. This has been done in order to make it easier to communicate large data sets between the components and to act as a common interchange format. But also, this central database makes it possible for a given repair session to persist in a convenient form over a long period of perhaps several months, if necessary. Information stored in the repository for a given session includes:

- all the data imported from the source checklists;
- any new checklists created as a result of merging checklists;
- all the conflicts that have been detected;
- the current state of each of the potential repairs;
- references to appropriate repair actions for each kind of conflict; and
- author abbreviation lists.

This information is used to store an entire session so that the user can return to the LITCHI software as information is obtained from taxonomic experts, in order to continue the repair process.

5.2. Conflict typing to reduce complexity

One of the problems encountered when trying to support constraint repair is that, for any given constraint, there will in general be a large number of repair updates which can

restore its consistency. For example, the constraint given earlier that no name may appear as both an accepted name and synonym in the same checklist can be repaired by any of the following actions:

- Change the conflicting synonym to something else;
- Change the conflicting accepted name into something else;
- Delete the synonym from the conflicting taxon in which it occurs;
- Delete the accepted name from the taxon in which it occurs, and promote one of its synonyms to be the new accepted name;
- Demote the conflicting accepted name to the role of synonym, and promote one of the synonyms to be the new accepted name;
- Merge the two taxa together to form a single taxon and delete the conflicting synonym; or
- Merge the two taxa together to form a single taxon and choose one of the non-conflicting synonyms to act as the accepted name of the new taxon.

In fact, we have given only a selection of the many possible repairs for this constraint⁷. According to the information present in the integrity constraint itself, these are all valid repairs. However, there may be additional contextual information about the conflict, that is not referred to in the constraint but which can be useful in reducing the number of repair options presented to the user. For example, the constraint does not distinguish between situations when the duplicated accepted name and synonym refer to the same taxon, and situations when they refer to different taxa. And yet quite different repair updates are appropriate in each of these two cases. Most significantly, for example, there is little value in providing the option to merge the taxa involved in the conflict if in fact only one taxon is implicated in it. Similarly, deletion of the duplicated synonym is a more suitable repair for cases when only one taxon is involved than when two are involved, since it does not result in the loss of any information. (Deletion of the synonym from a separate taxon means that the fact that that name was ever associated with that taxon is lost. In general, repairs which preserve the information content of the checklist are preferred over those which do not.)

In LITCHI, therefore, we have found it convenient to partition the conflicts relating to a particular constraint,

⁷When the constraint is represented in terms of our current schema, there are 255 repair updates that are possible theoretically, although many of these violate referential (or other) integrity and many of the rest do not represent a meaningful change to a checklist.

based upon further contextual information. Each constraint therefore has a number of associated conditions which determine the type of each violation. For example, the constraint given earlier:

$$\begin{aligned}
 &(\forall n, a, l, c_1, c_2, t_1, t_2)(\text{accepted_name}(n, a, c_1, l, t_1) \\
 &\quad \wedge \text{synonym}(n, a, c_2, l, t_2) \\
 &\quad \Rightarrow \text{pro_parte}(c_1) \wedge \text{pro_parte}(c_2))
 \end{aligned}$$

has two associated types, Type (i) and Type (ii), given by the conditions $t_1 = t_2$ and $t_1 \neq t_2$ respectively. The conflict detector applies these conditions to each violation found, to determine its type, and stores this information in the database. It can then be used to determine which set of repair options are appropriate for the constraint.

A repair update is inappropriate for a particular conflict type if the characteristics of that type violate the preconditions for the repair. For example, the `merge_taxa` (T1, T2) update has the precondition:

$$T1 \setminus == T2$$

Since this precondition cannot be satisfied by any type (i) conflict, it should not be presented as an option to the user. In order to improve the efficiency of the LITCHI system, details of which repair updates are appropriate to each constraint type have been manually inserted into a lookup table. However, it is easy to envisage the construction of some off-line tool which was able to create this table automatically, using the type predicates and the update preconditions.

5.3. The user interface

In order that the taxonomist can perform repairs effectively, a suitable user interface is needed. The user must be able to choose to build up his or her repair set in an arbitrary order, so that repairs which the taxonomist regards as easy to decide upon may be performed first, and other repairs can be delayed until after an expert has been consulted. Furthermore, in order to determine an appropriate course of action, the user must be able to browse each conflict in the context it occurs. For example, if a large number of taxa have been moved from one genus to another, this may have a bearing upon the genus with which the user considers it most appropriate to associate a given species. Another need is to allow the user freedom in the order in which (s)he answers questions associated with the choice of a suitable repair action. The LITCHI user interface therefore includes a conflict browsing screen, as illustrated in figure 3. This screen includes a hierarchical checklist browser on the left-hand side, with conflicting taxa highlighted by a radio button-like graphic. The panes on the right-hand side allow the

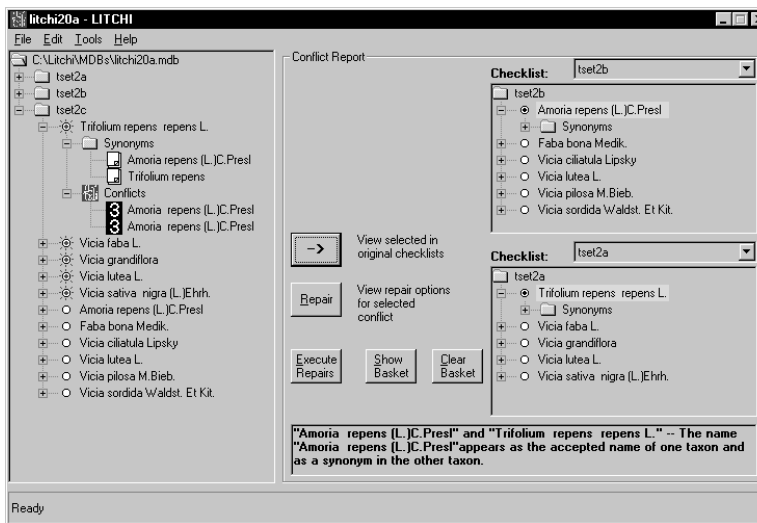


Figure 3. The LITCHI conflict browser

user to browse conflicts in the context of the original checklist(s) from which the current checklist was formed. The buttons in the middle of the screen allow the user to select a conflict for repair, in which case (s)he is presented with a screen displaying all the possible questions that could be answered first, and to browse the ‘basket’ of repairs so far accumulated.

6. Conclusions and future work

We have seen how, in order to implement a tool to meet our aim of enabling taxonomists to detect and resolve conflicts in taxonomic checklists, a number of techniques have had to be used, including techniques to deal with exceptions to conflicts; orthographic variation; long repair sessions; large numbers of possible repairs, and flexibility and availability of information during the repair process. In principle, one might anticipate that similar techniques will be required in other domains where one is seeking to obtain consistent nomenclature. If useful constraints cannot be expressed on the relationship between names themselves, however, it will be necessary to descend to the data level to identify conflicts in the classification by detecting conflicts in the data stored according to this classification. This is a challenging problem, unless either the data is entirely uniform in its semantics or there is sufficiently rich metadata to determine the relationships between data in heterogeneous systems.

We are currently working on how interactions between repair actions can be detected, so that if a user chooses to *remove* a repair from the repair set, it can be determined which of the other repairs selected are invalidated. One possibility for future work is to apply the techniques developed in LITCHI to other nomenclatural problems, as suggested

above. Another related problem is how to enrich metadata in a heterogeneous system to enable data-level conflicts to be detected. But although we distinguished carefully between our research and the research associated with systems such as ReTAX and HICLAS, we are attracted in the long term to the possibility of combining the constraints of LITCHI with the taxonomic consistency ideas of ReTAX, the explicit operators of HICLAS and the full taxonomic hierarchies of Kitakami *et al*, in order to build a tool that supports the entire process of taxonomic database merging.

7. Acknowledgement

The research reported in this paper was funded by a grant from the Bioinformatics committee of the BBSRC/EPSRC.

References

- [1] E. Alberdi and D. Sleeman. ReTAX: a step in the automation of taxonomic revision. *Artificial Intelligence*, 91:257–279, 1997.
- [2] F. Bisby, G. Russell, and R. Pankhurst, editors. *Designs for a Global Plant Species Information System*. Oxford University Press, Systematics Association, 1993. Special Volume No. 48.
- [3] R. Brummitt and C. Powell. *Authors of Plant Names*. Royal Botanic Gardens, July 1992.
- [4] E. Codd. *Providing OLAP (on-line analytical processing) to user-analysts: An IT mandate*. E.F. Codd and Assoc., 1993.
- [5] S. Embury, A. Jones, I. Sutherland, W. Gray, R. White, J. Robinson, F. Bisby, and S. Brandt. Conflict detection for integration of taxonomic data sources. In M. Oszoyoglu, editor, *11th International Conference on Scientific and Statistical Databases*, pages 204–213. IEEE Computer Society Press, 1999.

- [6] W. Greuter, F. Barrie, H. Burdet, W. Chaloner, V. Demoulin, D. Hawksworth, P. Jorgensen, D. Nicolson, P. Silva, P. Treharne, and J. McNeill. *International Code of Botanical Nomenclature*. Koeltz Scientific Books, 1994.
- [7] International Astronomical Union. Working group for planetary system nomenclature. In *Transactions of the International Astronomical Union*, volume 16B, pages 321–369, 1977.
- [8] S. Jung, S. Perkins, Y. Zhong, S. Pramanik, and J. Beaman. A new data model for biological classification. *Computer Applications in the Biosciences*, 11(3):237–246, 1995.
- [9] H. Kitakami, Y. Mori, and M. Arikawa. An intelligent system for integrating autonomous nomenclature databases in semantic heterogeneity. In R. Wagner and H. Thoma, editors, *7th International Conference on Database and Expert Systems Applications*, volume 1134 of *Lecture Notes in Computer Science*, pages 187–196. Springer, 1996.
- [10] S. Lapage et al., editors. *International Code of Nomenclature of Bacteria, 1990 Revision*. American Society for Microbiology, Washington, D.C., 1992.
- [11] I. C. on Zoological Nomenclature. *International code of zoological nomenclature*. International Trust for Zoological Nomenclature, London, 4th edition, 1999.
- [12] D. J. Orth and R. L. Payne. *Principles, Policies and Procedures: Domestic Geographic Names*. US Geological Survey Office of Geographic Names, 3rd edition, 1997.
- [13] C. Stace. *Plant Taxonomy and Biosystematics*. Edward Arnold, first edition, 1980.
- [14] I. Sutherland, S. Embury, W. Gray, A. Jones, F. Bisby, S. Brandt, J. Robinson, and R. White. Knowledge Integrity Testing for the Integration of Taxonomic Databases. In D. Gilbert, editor, *Proceedings of the Second Workshop on Constraints in Bioinformatics/Bioinformatics*, Pisa, Italy, Oct. 1998.
- [15] R. White and R. Allkin. A language for the definition and exchange of biological data sets. *Mathematical and Computer Modelling*, 16:199–233, 1992.
- [16] R. White and R. Allkin. A strategy for the evolution of database designs. In F. Bisby, G. Russell, and R. Pankhurst, editors, *Designs for a Global Plant Species Information System*, pages 284–303. Oxford University Press, Systematics Association, 1993. Special Volume No. 48.