Digital Look Up Tables and Real Number Theorem Proving

A. A. Adams
School of CS, Cyb and EE
University of Reading *

September 28, 2001

Abstract

We consider the utility of digital look up tables, as adjuncts/helpers to computer algebra systems. The requirements for dealing with logical side conditions raised by such tables are considered and proposals for using theorem proving technology as black box aids are considered. In addition, the use of real number theorem proving libraries to support validation of table entries is also presented.

1 Introduction

Mathematics is performed by a number of distinct groups. The distinguishing characteristics that are relevant to this paper are whether a user is involved in applying mathematics or in extending mathematics. Both kinds of user obviously have a role to play in the development of mathematical software. For centuries problems requiring long, involved calculations have been solved by turning to mathematical tables. This results in a much lower level of mathematical knowledge required of the calculator (the person who calculates). The calculator can use a look up table without any knowledge of how the answers in the table are calculated. It is possible, although generally not appropriate, that the calculator may not even understand the meaning of some of the concepts they are looking up, provided they understand how to use the tables. Even with the advent of pocket calculators, PDAs and laptop computers, engineers and scientists will still routinely refer to mathematical tables. The fact that the Reading University Library copy of [Bey87] is still borrowed for overnight or longer a number of times per year is testament to this fact. The differences between [CRC64] (the 2nd edition from 1964) and [Bey87] (the 28th edition from 1987) highlight the change from an emphasis on numeric and symbolic tables in the '64 edition to almost purely symbolic entries in the '87 edition. In time, computer assistance may make all paper based tables redundant, but not yet, as we shall demonstrate.

Computer Algebra Systems [CAS] are now described as "tools for all phases of technical computation" or "complete mathematics assistants". However, the Mathematica web site [Wol] does admit: "Have confidence in the accuracy of numerical results!' i.e. that you cannot really have confidence in the accuracy of its symbolic results. CAS such as Maple [Map] and Mathematica already use ad hoc look up tables to cover errors in their systems. Routine publication in the scientific literature of incorrect results from CAS leads to a future version of the software producing the correct answer: until you "tweak" the query, that is. Introducing a parameter in place of a concrete value or similar minor changes to the query

*This work was supported by the UK EPSRC Grant GR/L48256 and by the EU Grant Calculemus RTN1-1999-00301.

often produces the original incorrect result once again. The implication of this, which has occasionally been confirmed by those working for the CAS producers, is that a look up table has been implemented to catch known bugs before they're subjected to the flawed algorithms.

Some efforts have also been made to digitise existing mathematical tables. CRC have produced a CDRom version [Zwi98] of their [Zwi96] (of which 30 printed editions have appeared). Unfortunately, the CDRom version has a number of flaws, despite its inclusion of a Maple kernel as an enhancement. Many of its entries are stored as graphics, which is not helpful in allowing search, nor in copying and pasting into the Maple kernel or any other mathematical or typesetting software.

These issues do not lead to a conclusion that computer-based versions of the mathematical tables are worthless. On the contrary, the idea is an excellent one. The problems highlighted above simply show that a more rational approach needs to be taken, considering the issues of calculation and look up on a higher level, while considering the underlying technologies needed to produce good working mathematics assistants, building on the good work of the past and learning from the mistakes that have been made.

This paper will not lay down a step-by-step path towards this goal. Instead, it is meant to raise some of the issues and discuss some possible solutions, contrasting mutually exclusive approaches and discussing the benefits of merging as many as are compatible.

1.1 Outline

To understand digital tables, one must first understand the uses and abuses of paper tables, so section 2 contains an overview of the books of tables that reside in almost all mathematics libraries. Following on from this section 3 covers the principle background to digital tables, first in 3.1 presenting material on digitised versions of existing tables and then in 3.2 presenting the existing work on creating new tables, including references to the author's previous work in this area.

Having presented the background to digital look up tables, the technical issues at the forefront of this work are then considered: focussing on the language of table entries in 4 and the encoding and semantics of that language in 4.1, with a focus first on normalisation (4.2) and equivalences (4.3) of the mathematical expressions, followed by a discussion of storage, retrieval and matching in 4.4.

In the final two sections, some concrete implementation issues are discussed, such as the architecture of the table, and interoperability with existing mathematical software, in section 5. A conclusions section (6) rounds off the paper, drawing all these threads together again

2 Background: Paper Tables

While all readers of this paper may have looked something up in a set of paper based tables at some point, they may or may not have used them extensively, so in this section the main relevant points of paper based mathematics tables are presented.

Books of numeric tables have been virtually replaced by computers and the pocket calculator. Four or six figure tables of logs, trig functions etc. such as [MTC31] are now primarily of interests to science historians rather than tools in daily use by scientists in all subjects.

Symbolic tables such as [GR65, Zwi96] are still of use, although their utility is gradually being replace by computer-based tools, as described above. In considering how best to replace such books of tables, consideration of existing paper tables is useful.

We will consider tables of integrals as an example of the types of query that may be answered using paper based tables, since this is the area of application of the existing digital tables we consider below.

The main consideration of paper based tables that we wish to highlight is the look up and identification procedure. Say we wish to integrate:

$$\int_0^{\frac{\pi}{2}} \frac{1}{1+\sin(2x)}\, dx$$

To look this up in [Bey87], we find that the section on integrals is subdivided for various polynomial forms, but has only a single subsection (of 12 pages) dealing with "Forms Involving Trigonometric Functions". This subsection contains 151 separate entries, which we must systematically check through until we find entry 337:

$$\int \frac{dx}{1 \pm \sin(ax)} = \mp \frac{1}{a} \tan\left(\frac{\pi}{4} \mp \frac{ax}{2}\right)$$

Even in this well-respected book of tables there is no indication of the limits of applicability of this indefinite integral to definite integrations. Our query above has no problems, but it requires a fairly good insight into the nature of the sin function to realise this.

Having found our indefinite integral, we still have much work to do in matching the values of our query to the table entry, substituting them in the result, and then performing further calculation on that result to find that the answer is actually 1.

3 Background: Digital Tables

There have been a number of projects looking to provide digital look up tables. Most have been research projects considering how to produce useful tables for a specific domain. There has also been work done on the idea of digitising existing tables. These different aspects are considered separately below.

3.1 Digitising Existing Tables

The attempt by CRC to digitise their existing paper table [Zwi96] on a CDRom [Zwi98], and to include a Maple kernel for further computation, was an interesting experiment. As mentioned above, however, it is a deeply flawed resulting product. Since a large minority of the entries are stored simply as graphics, there is no possibility of performing automated search or substitution, no ability to transfer the entries to the Maple kernel for computation, and no ability to re-size the text for easier viewing. About as useful as scanning a novel as a set of graphics images to try and replace paper books with e-books.

A similar project [GR96a] digitising [GR96b] has been heavily criticised by Fateman, in an unpublished article, on various points. The main point to note here is that the entries are primarily stored as TEX, and thus again in a form suited for display rather than further symbolic calculation. A new edition [GRJZ00] of the paper version has been produced, but there are no signs of a new edition of the CDRom.

The shortcomings of these projects are even more disappointing given that research into optical recognition of mathematical symbols has been performed (for example [FTBM96]), with specifically this sort of digitisation in mind. The recent development of OpenMath [Dew00], a specification of an interchange language for mathematical software packages, might provide solutions to some of the problems of these previous digitisation projects.

However, the general approach taken would not seem that successful. In order to make digitised tables useful they must address needs beyond the utility of a printed book.

3.2 Creating New Digital Tables

Various projects have produced limited tables from scratch. The source of material for these tables may well be the contents of printed paper copies, but simple digitisation of them was not the intent of such work. Rather, these projects have investigated various aspects of the problem of digital tables. This paper, in part, is an attempt to bring some of these aspects together in a coherent picture.

We will focus on three particular projects in this section, although doubtless there are other projects in existence focussing on other aspects of digital maths tables.

The projects on which we shall focus our attention are:

1. Einwohner and Fateman's TILU [EF95, FE]
2. Dalmas *et al*'s MFD2 [DGH96, Huc97]
3. Adams *et al*'s DITLU [AGLM99b, AGLM99c]

All of these tools were designed and implemented as research tools to investigate aspects of the problem rather than as production tables for distribution. The code for each of them is probably available from the authors or is available as an on-line supplement to published papers.

3.2.1 TILU

TILU stands for *Table of Integrals Look Up.* An implementation has been available for accessing via the web [FE], although this service may have been terminated (as of writing the service is unavailable).

Einwohner and Fateman's work (as exposited in [EF95] focussed primarily on the practicalities of the search problem for large tables of mathematical formulae, using integrals as a specific case study. Their primary approach involves the use of hash table look up (to avoid a trade-off between size of table and search time) and successive approximations via expression kernel identification (to increase the speed of identification of possible matches). None of their techniques appear to be incompatible with other work such as MFD2 or DITLU. As with most other work in this area, little attention is given to pre- and post-processing of the query or answers, concentrating on the still under-researched area of how to implement efficient and useful digital look up tables at their core.

The system of "successive approximate matches" categorises a query in successive rounds, and then attempts to match the query against the categorisations of table entries. The example given in [EF95] is:

> Thus, e.g., the integrand $1/(x^2+1)$ would match $1/(x^2+1)$, $1/(x^2-a)$, $1/(x^2+a^2)$ and $(x^2+bx+c)^{-1}$ (etc).

They continue to describe the matching process with the categorisations made explicit:

> The keys [associated with the integrand] grow in discriminatory power and length as one goes down the expression tree of the integrand. In the above example, the first two keys would be the atom `reciprocal` and the list `(reciprocal quadratic)`.

The main thrust of the TILU was to demonstrate the possible power of this successive matching approach, and little work was done on further aspects of unification, and parameter constraint, the focus of the other two systems we will now consider.

One specific limitation to note about the TILU is that the definite integrals contained in the table were of a very specific form and highly limited in their limits of integration (only about eight different limits were in the table).

3.2.2 MFD2

MFD2 (Mathematical Formula Database 2) is a more wide-ranging project than the other two presented in this section. The aim of MFD2 is to consider a completely generic mathematical formula database. As such, the specific issues of integration are not a concern, simply the general issues with matching constrained parametric mathematical expression. The interface of MFD2 includes two mode of operation. The first is for a user to check if a proposition appears in the database. This proposition can be an equality, inequality or any other mathematical proposition, such as irreducibility. So long as the predicate of interest is present in the database, an answer to the question may be contained therein. The other mode of usage proposed for MFD2 is that of calculation, where an expression is entered and a matching right hand side of an equality is searched for. This latter format is used as a substitute for such queries as are the primary mode for TILU and DITLU.

The central idea of MFD2 which sets it apart from other systems is that it uses logic programming ideas to aid unification in the presence of constrained parameters. We will consider this idea in more detail in section 4.4, but present it here as implemented in MFD2.

Entries in MFD2 may have constraints on the parameters contained within them. Thus, the example from the abstract of [Huc97]:

$$x > 1 \Rightarrow \ln(x) > 0 \tag{1}$$

So, a query to the database:

$$a > 1 \Rightarrow \ln(a^2) > 0 \tag{2}$$

matches x to a^2, and then launches a logic programming search for evidence that:

$$a > 1 \Rightarrow a^2 > 1 \tag{3}$$

which fact should be in the definitions of $<$ and ^ as held in the logic program. This sequence can be extended to arbitrarily complicated expressions, with the logic program regarded as a search mechanism for existence proofs of values which allow the parametric constraints to be satisfiable. The TILU contains something similar as a method of matching multiple candidates from the table to a particular query, although the TILU version was rather rudimentary in comparison.

3.2.3 DITLU

The DITLU (Definite Integral Table Look Up) is a prototype system which drew upon the ideas of both the TILU and the MFD2. The aim of the DITLU is to sit within a CAS and provide known answers to parametric definite integration problems. This extended the work done by the TILU to cover a much wider range of definite integrals. Indefinite integration within computer algebra packages is quite a well understood problem, although efficient solutions in the presence of multiple parameters are still under development. A prototype implementation of the DITLU was produced, programmed in Lisp, and a more robust implementation inside a CAS is currently under consideration. The main focus of this work was on rationalising and extending the consideration of parametric constraints compared to that of the MFD2. While the MFD2 used a logic program to perform both the matching and constraint checking, the prototype DITLU included a separate unification routine which passed certain parts of the problem off to a constraint checker which uses the theorem prover PVS as its logic engine. The advantage of such a system is that the dependence on a complicated special-purpose logic program is replace with dependence on a general purpose theorem prover. The standard architecture is similar, although the unification and search routines are designed such that they may be re-implemented using ideas from the prototype but including methods from the TILU and other sources.

Another interesting aspect of the DITLU is the concept of validating the table entries using the same technology in PVS as that used to check parameter constraints. It is well known [KG68, EF95] that existing look up tables (both paper and digital versions) contain errors. Such errors could be minimised, if not entirely avoided, were table entries to be validated. This validation, since it would only have to be performed once, could be manually directed by experts in the field (here definite integration) rather than requiring the CAS user to perform proofs in an unfamiliar system or requiring highly intelligent automated proof systems.

The kinds of query and process used in the DITLU are shown in the following example.

Say the table contains the following definite integral:

$$\int_a^b \frac{1}{x+c}\,dx = \begin{cases} \ln|b+c| - \ln|a+c| & \text{for } (a<c \wedge b<c) \vee (a>c \wedge b>c) \\ \text{Unknown} & \text{for } a=c \vee b=c \vee \\ & \quad (a<c \wedge b>c) \vee (a>c \wedge b<c) \end{cases}$$

The query: $\int_2^3 \frac{1}{x+1}\,dx$ will be transformed into the parametrised query

$$\int_l^m \frac{1}{x+k}\,dx,\ l=2,\ m=3,\ k=1$$

and matched with the table entry. The following queries will then be sent to the PVS theorem prover:

$$\begin{array}{ll} Q_1 & \neg\exists l,m,k:\mathbb{R}.l=2 \wedge m=3 \wedge k=1 \wedge \\ & ((l<k \wedge m<k) \vee (l>k \wedge m>k)) \\ Q_2 & \neg\exists l,m,k:\mathbb{R}.l=2 \wedge m=3 \wedge k=1 \wedge \\ & (l=k \vee m=k \vee (l<k \wedge m>k) \vee (l>k \wedge m<k)) \end{array}$$

See [AGLM99a] for an explanation of the reason for attempting to prove the negation of the constraints. From an examination of these logical queries it is obvious that Q_1 is false while Q_2 is not only true, but easily provable. The DITLU will therefore return the answer:

$$\ln|3+1| - \ln|2+1|$$

which may then be subjected to further simplification.

4 Mathematical Expression Languages

One of the problems involved in developing digital look up tables, either as part of a larger mathematical software package or as a stand-alone tool, is the issue of the interface language. mathematical notation developed over many centuries of hand-written notations, with movable-type printing and wider distribution of mathematics texts causing convergence of notation. The notation of mathematics is still a very rich language, however, in terms of the number of symbols available in it, and the overloading of those symbols for multiple areas of interest.

The advent of machines, first typewriters and then computers, into the world of mathematics caused quite a few problems, suddenly limiting the notations available. Quite a number of maths and logic texts from the thirties through even to the eighties were typewriter written, with the extra maths symbols written in by hand before photographic copying techniques were applied for printing.

Knuth's $\TeX$ was a great leap forward in allowing for the typesetting of mathematics by the author, without relying on the expertise of a movable type operator or tedious hand-addition of symbols. However, the problem of using a standard keyboard to interface with a computer assistance tool for maths is still under consideration.

Various projects have looked at this problem, and various languages have evolved to cope. In recent years, the problem of interoperability of different forms of maths assistance (multiple computer algebra packages, theorem proving environments, display environments) has been addressed by the OpenMath project [Dew00]. The first results of this project have proved very useful in terms of display and interoperability between systems: passing messages between any two mathematics packages now involves having OpenMath compliant interfaces for each rather than writing a purpose-built translator. Even the problem of semantics has been addressed with the use of content dictionaries fixing the relative meanings of symbols within a context.

The OpenMath approach presents solutions to some of the problems involved in developing digital look up tables, but not all. The question of the language in which one stores a large table is obviously OpenMath, should one wish to develop a table accessible to many different tools. The problem of how to pass different parts of the look up process to tools with different capabilities is also solved by working in an OpenMath-compliant object language.

The user-interaction with a table will almost certainly be within an ASCII text environment, such as those deployed within the various theorem proving and CAS available now. Indeed, it is unlikely that direct access to the table would be required, since a table is unlikely to include any calculation of answers from the generic form of their storage. More commonly a CAS will be the front-end presented to a user, with the table and a theorem prover used as black box subsidiary programs. An interface for the compiler/maintainer of the table might be useful, although it is likely to remain useful to access these functions via a CAS also.

4.1 Semantics and Encoding

As mentioned above, the language used in the table can be represented using the OpenMath standard. This has a number of advantages, not least that a single table might then be accessible to a number of different applications. There still remains the problem of specifying the semantic of the language, and restricting the language to a set of symbols.

Following the OpenMath approach, it is obvious that the table should be so constructed as to allow the same mechanisms to be used for various application areas within mathematics. For instance, in real analysis, both addition ($+$) and multiplication ($\times$) are associative-commutative, whereas in matrix algebra multiplication is non-commutative.

Each area might then have its own content dictionary, in which the properties of the various operators are described (such as associativity and commutativity) may be stored. The other mechanisms of table creation, maintenance and retrieval can then use these properties as necessary. In the DITLU, for example, aspects of this were present in the implementation. An ordering of the function symbols was defined in a separate global variable to allow for addition/removal of function symbols from the mathematical language of the system. The AC properties of $+$ and $\times$ was also defined separately and their interactions depended separately on their common distributivity axiom and their own AC properties, rather than being hardwired into the normalisation routines.

4.2 Normalisation

Before considering how one store or retrieves table entries, it is necessary to consider normalisation routines. While there appear to be no fully normal forms for algebraic expressions such as appear as integrands in [Zwi96], some useful normalisation can still be performed on such expressions, which can improve the efficiency of matching algorithms, both in terms of time taken to find a match or lack of a match, and also in terms of finding all possible matches. The author is unaware of any categorisations of such a language with respect to its normalisation or unification properties. Searches of the literature on unification (e.g. [Kir90]) reveal that most work in this area depends on very limited sets of function, with highly constrained properties. The intuition gained from consideration of the work that does exist on unification in the presence of AC operators and on operators with units is that unification is undecidable and normalisation likewise generally unachievable with such a rich language as generic maths look up tables require. Certain restricted subject areas might lend themselves to rigorous normalisation and matching, but certainly the language of real analysis is too rich. As discussed in [Ada], normalising expressions for storage in a table, and for queries before matching is attempted can greatly increase the chances of matching queries to entries. The general problems of AC operations, units and inverses are added to by the plethora of equivalences present in real analysis. It is possible that theorem proving support even in the normalisation routines might be beneficial. This concept will be discussed further in section ??.

4.3 Equivalence

As mentioned above, one of the main problems with the lack of a normal form for the sorts of expressions we wish to store and use as queries, is the large set of equivalences. As mentioned in [Ada], rewriting orientations imposed on a subset of the equivalences is useful. It is not a complete set of equalities, however, and orientation of even unconditional equivalences fails to produce a normalising set of rules. Consider, for instance, some of the trigonometric identities commonly used in performing geometric proofs:

$$\sin^2(t) + \cos^2(t) = 1 \tag{4}$$

$$\sin(u+v) = \sin(u)\cos(v) + \cos(u)\sin(v) \tag{5}$$

$$\sin(2u) = 2\sin(u)\cos(u) \tag{6}$$

A properly developed theory of transcendental functions within a theorem proving environment should include most or all of these identities as theorems, but the use of these theorems to match two possibly equivalent expressions requires much more than simply the existence of the theorems. At some point, the problem becomes undecidable, as evidenced by explorations of the "zero constant" problem [RF94]. Artificial Intelligence approaches to such problems have also been investigated [SBB+89].

4.4 Storage, Retrieval and Matching

Given the number of entries in existing paper tables, even given that the DITLU contains approximately ten times as much information per entry as [GR65], the volume of data itself should not cause any real problems for information processing, provided an adequately focussed retrieval mechanism is used. Einwohner and Fateman [EF95] were entirely correct in that a look up table which scans every entry for a match to a query would rapidly become a severe drain on the computing resources available. Their approach, of expression kernel indexing, could be easily combined with the more advanced matching capabilities such as those prototyped in MFD2 and DITLU. This would provide a swift but accurate identification of possible matching table entries, but still leave the details of matching the user's requirements with the table entries to a sophisticated logical process.

As mentioned previously, it would seem probable that an interchange language such as OpenMath would be the best format for storing entries, allowing multiple systems to easily contribute to and access the actual database. Completely separate programs might then be used for creating and maintaining the table compared to accessing the information contained therein. More discussion of this point is contained in the section 5 below.

We have previously briefly discussed the approach of using logical support for the matching process, either in the form of a logic program (MFD2) or a theorem prover (DITLU). Both approaches have merit, and their use is not, in fact, exclusive (see section 5). In this section we present a more detailed description of this concept, abstracted away from the specifics of MFD2 or DITLU.

When attempting to match constrained parametric expressions, we are performing a more complex task than purely syntactic unification. Each parameter has a range of possible values which may be assigned to it. Provided the general structures of the two expressions match then bindings between the parameters may be more complex than $a \to p,\ b \to q,\ c \to r$, the norm in first order unification. nor is it higher order unification that we are really interested in, since our first order function symbols are fixed and must match. Instead, what we are really interested in is a constraint satisfaction problem. It is, however, a constraint satisfaction problem which is so far outside the scope of the usual approaches to CSPs as to require new approaches. Standard CSPs usually restrict their attention to linear or quadratic functions, occasionally adding in one different function or a higher order polynomial. Even there, the problem quickly becomes unsatisfiable. In order to accommodate these aspects of the matching required by a digital look up table, the following process would appear to be suitable. In the following, parameter is taken as any name that is not a recognised function symbol, a recognised constant symbol or the variable of integration. The variable of integration is always assumed to be "x".

1. Rewrite expressions to:
   - replace binary minus with addition and unary minus;
   - replace binary occurrences of AC operators with multiary function applications;
   - replace division by multiplication and unary negation;
   - perform various distributions of operators to ensure a normalisation effect.
2. Identify sub-expressions which do not contain the variable of integration, and where they occur as arguments of multiary applications of AC operators, commute their positions together, e.g:

$$\sin(a+2x+1)+\sin(x)+\cos(x)+\sin(a)+\cos(2a)+x+a+b+2x^2 \tag{7}$$
$$\to$$
$$\sin(2x+a+1)+\sin(x)+\cos(x)+x+2x^2+\sin(a)+\cos(2a)+a+b \tag{8}$$

3. Replace these "parametric constant" sub-expressions with one new parameter each and add equality constraints.

$$\sin(2x+a+1)+\sin(x)+\cos(x)+x+2x^2+\sin(a)+\cos(2a)+a+b \tag{9}$$
$$\to$$
$$\sin(t_1x+t_2)+\sin(x)+\cos(x)+x+t_3x^{t_4}+t_5 \tag{10}$$
$$t_1=2 \qquad t_2=a+1 \qquad t_3=2 \qquad t_4=2 \qquad t_5=\sin(a)+\cos(2a)+a+b$$

4. Merge "new parameters" which have identical values. Note that no processing is done here, a simple syntactic equality is applied. This is a small heuristic to improve speed

later, but spending logical processing power on this would not be worthwhile. In our example we "merge" t_3 and t_4 into t_1:

$$\begin{aligned} & \sin(t_1 x + t_2) + \sin(x) + \cos(x) + x + t_3 x^{t_4} + t_5 \quad (11) \\ t_1 = 2 \qquad t_2 = a+1 \qquad t_3 = 2 \qquad t_4 = 2 \qquad & t_5 = \sin(a) + \cos(2a) + a + b \\ & \to \\ & \sin(t_1 x + t_2) + \sin(x) + \cos(x) + x + t_1 x^{t_1} + t_5 \quad (12) \\ t_1 = 2 \qquad t_2 = a+1 \qquad & t_5 = \sin(a) + \cos(2a) + a + b \end{aligned}$$

5. Rewrite multiary applications of AC operators to group sub-expressions containing the same top-level function symbol.

$$\sin(t_1 x + t_1) + \sin(x) + \cos(x) + x + t_1 x^{t_1} + t_5 \tag{13}$$

$$t_1 = 2 \qquad t_2 = a+1 \qquad t_5 = \sin(a) + \cos(2a) + a + b$$

$$\to$$

$$(t_1 x^{t_1} + x + t_5 + (\cos(x) + (\sin(t_1 x + t_2) + \sin(x)))) \tag{14}$$

$$t_1 = 2 \qquad t_2 = a+2 \qquad t_5 = \sin(a) + \cos(2a) + a + b \tag{15}$$

This is done in a pre-determined order, in this case putting "polynomial" expressions (ordered according to the exponent where it is a constant value) first followed by cos then sin etc. Precisely which ordering would give the most efficient matching procedures requires further research.

The purpose of this rewriting is to separate out the two aspects of the problem: matching structure and matching parameters. parameters now occur singly, with side conditions giving their values in terms of the originally input parameters. Matching the structure of the expression held in a table to the structure of a query is now easier, but leaves behind the question of satisfying the sets of constraints that result. Note that these equality constraints are added to the branching constraints" giving rise to particular answers. On the whole, a rational approach to building a table would involve table entries which already fit the above properties before being submitted. This will produce cleaner tables, although branching side conditions might be made more complicated by this process.

For the sake of argument, however, consider

$$3 + 2x^2 + \sin(x) + \cos(x) + x + \sin\left(2x + \frac{\pi}{2}\right) \tag{16}$$

as a query posed to a table containing (14). The query (16) will be re-written to:

$$\left(t_6 x^{t_6} + x + t_7 + (\cos(x) + (\sin(x) + \sin(t_6 x + t_8)))\right) \tag{17}$$

$$t_6 = 2 \qquad t_7 = 3 \qquad t_8 = \frac{\pi}{2} \tag{18}$$

Expressions can then be unified using the following procedure:

1. Check the top-level function symbols are the same.
2. If the top-level function symbols are the same AC operator, try to match all possible binary combinations of arguments.
3. If the top-level function symbols are the same, try to match each pair of arguments in the same positions.

4. Parameters and the variable of integration are the only "atomic" expressions. Any attempt to match a functional expression with an atomic expression should cause a failure, as should any attempt to match the variable of integration and a parameter.

5. Two parameters (say t_i and t_j) always match, and a new constraint is added ($t_i = t_j$).

So, for our example, expressions (14) and (17) will provide a suitable candidate match because the sub-expressions:

$$(\sin(t_1 x + t_2) + \sin(x))(\sin(x) + \sin(t_6 x + t_8))) \tag{19}$$

have a top-level AC operator (+) and therefore each binary pairing of the arguments is checked for a match. We are then left with a set of constraints (C), with three sources: (15), (18) and the constraints produced in the matching process:

$$\begin{gathered} t_1 = 2 \qquad t_2 = a + 2 \qquad t_5 = \sin(a) + \cos(2a) + a + b \\ t_6 = 2 \qquad t_7 = 3 \qquad t_8 = \frac{\pi}{2} \\ t_6 = t_1 \qquad t_6 = t_1 \qquad t_7 = t_5 \qquad t_6 = t_1 \qquad t_8 = t_2 \end{gathered} \tag{20}$$

Note the multiple occurrence of $t_6 = t_1$ caused due to separate instances of the parameters occurring in the same positions.

In a full look up procedure the set of constraints C would be added to each set of constraints for particular answers to the integral (along with the matching constraints for the limits of integration). If there exist values for the original parameters (a and b) which allow this set of equalities to be satisfied (together with any initial constraints on the parameters), then we should consider the outcome indicated by that set of constraints. De-referencing of all expressions back to the original parameters must also be performed before returning answers to the user.

The complicated constraint satisfaction problems such as we encounter here cannot be solved by existing CSP methods, which are confined to linear and quadratic constraints. Even cylindrical algebraic decomposition is not sufficient to the task, since the constraints are not purely algebraic (they may contain transcendental functions). Thus an approach such as that of PRESS [SBB+89] may be fruitful. The full power of a theorem proving environment such as HOL or PVS is obviously the correct milieu in which to develop such automated proofs. Some early efforts in this direction may be found in [Got00].

5 Architecture: Interoperability

As mentioned previously, it is fairly obvious that the effort in producing a verified look up table is sufficient that it is worth producing the table in a format which is accessible via a number of systems. The recent development of OpenMath would seem an appropriate aid to making tables interoperable for different systems. The question of the environment in which the table might be developed is still open, however. There are a number of existing systems which have OpenMath-compliant interfaces, and more are being developed.

Some of the pre- and post-processing facilities of CAS such as Mathematica and Maple may be of use in developing digital look up tables. However, a fair number of the problems of using CAS to perform definite integration lies in these very algebraic processing facilities (see [AGLM99a, ?] for details of these problems). Given this, it might be just as well to produce the implementation of the table as either a stand alone program or within the environment of a theorem prover. This latter approach has the advantage that an interface to the theorem prover such as that described in [?] between Maple and PVS, is not needed to provide the linkage between the table and the theorem prover.

Whatever the implementation environment of the table, it is obvious that storing table entries in an OpenMath-compliant language is appropriate. Indeed, the XML which comprises OpenMath at base, is an entirely appropriate format for what is, effectively, a database.

6 Conclusions

We have considered the utility of tables of mathematics, which we believe is still useful despite the development of general mathematics assistants. Even if no other justification is needed, the fact that it is much easier to verify the contents of a mathematical table as opposed to to the algorithms in a computer algebra system, is sufficient. Given appropriately efficient look up and search procedures, such look up can even be faster than calculation in many cases. Certainly in the case of parametric definite integrals, the complexity of calculating all the branches each time is much greater than that needed to consider the branch pruning suggested for the DITLU. The continued production of paper tables, and the attempts at digitising those tables, demonstrates the interest still in such reference tools.

We have considered many of the complications and necessary technologies for such tables in this paper, and further work in the area is expected to continue. In particular, work on the theorem proving support necessary to verify tables is ongoing.

References

[Ada] A. A. Adams. Theorem Proving in Support of Computer Algebra — DITLU: A Definite Integral Table Lookup. In preparation for journal submission.

[AGLM99a] A. A. Adams, H. Gottliebsen, S. A. Linton, and U. Martin. A Verifiable Symbolic Definite Integral Table Look-Up. Technical Report CS/99/3, University of St Andrews, 1999. http://www-theory.cs.st-and.ac.uk/publications/CAAR/CS993.

[AGLM99b] A. A. Adams, H. Gottliebsen, S. A. Linton, and U. Martin. Automated theorem proving in support of computer algebra: symbolic definite integration as a case study. In [Doo99], 253–260.

[AGLM99c] A. A. Adams, H. Gottliebsen, S. A. Linton, and U. Martin. VSDITLU: a verifiable symbolic definite integral table look-up. In [Gan99], 112–126.

[Bey87] W. H. Beyer, editor. *CRC Standard Mathematical Tables.* CRC Press, 28th edition, 1987.

[CL96] J. Calmet and C. Limongelli, editors. *Design and Implementation of Symbolic Computation Systems, International Symposium, DISCO '96.* Springer-Verlag LNCS 1128, 1996.

[CRC64] CRC (Chemical Rubber Company), editor. *Handbook of Mathematical Tables.* CRC Press, 2nd edition, 1964.

[DGH96] S. Dalmas, M. Gaëtano, and C. Huchet. A Deductive Database for Mathematical Formulas. In [CL96], 287–??

[Dew00] M. Dewar. Special Issue on OPENMATH. *ACM SIGSAM Bulletin*, 34(2), June 2000.

[Doo99] S. Dooley, editor. *Proceedings of the 1999 International Symposium on Symbolic and Algebraic Computation.* ACM Press, 1999.

[EF95] T. Einwohner and R. J. Fateman. Searching techniques for Integral Tables. In [Lev95].

[FTBM96] R. Fateman, T. Tokuyasu, B. P. Berman, and N. Mitchell. Optical Character Recognition and Parsing of Typeset Mathematics. *Journal of Visual Communication and Image Representation*, 7(1):2–15, March 1996.

[FE] R. J. Fateman and T. Einwohner. Tilu table of integrals look up. Web Service. http://torte.cs.berkeley.edu:8010/tilu.

[Gan99] H. Ganzinger, editor. *Automated Deduction — CADE-16*. Springer-Verlag LNAI 1632, 1999.

[Got00] H. Gottliebsen. Transcendental Functions and Continuity Checking in PVS. In [HA00], 198–215.

[GR65] I. S. Gradshteyn and I. M. Ryzhik. *Table of Integrals, Series and Products*. Academic Press, 1965.

[GR96a] I. S. Gradshteyn and I. M. Ryzhik. Table of Integrals, Series and Products. CD-Rom, Academic Press, 1996.

[GR96b] I. S. Gradshteyn and I. M. Ryzhik. *Table of Integrals, Series and Products*. Academic Press, 5th edition, 1996.

[GRJZ00] I. S. Gradshteyn, I. M. Ryzhik, A. Jeffrey, and D. Zwillinger. *Table of Integrals, Series and Products*. Academic Press, 6th edition, 2000.

[HA00] J. Harrison and M. Aagaard, editors. *Theorem Proving in Higher Order Logics: 13th International Conference, TPHOLs 2000*. Springer-Verlag LNAI 1869, 2000.

[Huc97] C. Huchet. *Bas de donées pour les relations mathématiques*. PhD thesis, Université de Nice – Sophia Antipolis, 1997.

[Kir90] C. Kirchner, editor. *Unification*. Academic Press, 1990.

[KG68] M. Klerer and F. Grossman. Error Rates in Tables of Indefinite Integrals. *Journal of the Industrial Mathematics Society*, 18:31–62, 1968.

[Lev95] A. H. M. Levelt, editor. *Proceedings of the 6th International Symposium on Symbolic and Algebraic Computation, ISSAC '95*. Springer-Verlag LNCS 1004, 1995.

[Map] www.maplesoft.com.

[MTC31] L. M. Milne-Thomson and L. J. Comrie. *Standard four-figure mathematical tables*. Macmillan, London, 1931.

[RF94] Dan Richardson and John Fitch. The identity problem for elementary functions and constants. In *ISSAC '94: Proceedings of the 1994 International Symposium on Symbolic and Algebraic Computation: July 20–22, 1994, Oxford, England, United Kingdom*, 285–290. ACM Press, 1994.

[SBB+89] L. Sterling, A. Bundy, L. Byrd, R. O'Keefe, and B. Silver. Solving symbolic equations with PRESS. *J. Symbolic Comput.*, 7(1):71–84, 1989.

[Wol] www.wolfram.com.

[Zwi96] D. Zwillinger, editor. *CRC Standard Mathematical Tables and Forumlae*. CRC Press, 30th edition, 1996.

[Zwi98] D. Zwillinger. Standard math interactive. CD-ROM, 1998.