Automated theorem proving in support of computer algebra: symbolic definite integration as a case study

A. A. Adams H. Gottliebsen S. A. Linton U. Martin *

Department of Computer Science,
University of St Andrews
{aaa,hago,sal,um}@cs.st-and.ac.uk

Abstract

We assess the current state of research in the application of computer aided formal reasoning to computer algebra, and argue that embedded verification support allows users to enjoy its benefits without wrestling with technicalities. We illustrate this claim by considering symbolic definite integration, and present a verifiable symbolic definite integral table look up: a system which matches a query comprising a definite integral with parameters and side conditions, against an entry in a verifiable table and uses a call to a library of lemmas about the reals in the theorem prover PVS to aid in the transformation of the table entry into an answer. We present the full model of such a system as well as a description of our prototype implementation showing the efficacy of such a system: for example, the prototype is able to obtain correct answers in cases where computer algebra systems [CAS] do not. We extend upon Fateman's web-based table by including parametric limits of integration and queries with side conditions.

1 Introduction

Computer aided theorem proving means the use of computers to produce formal proofs in a given logical system: a practical version of Hilbert's programme. It ranges from implementations of decision procedures, semi-decision procedures, and collections of strategies for the machine to try which will not necessarily find a proof, to programs which check line by line whether or not the input generated by the user or another program is a valid proof. The rise of formalism in computer science since the mid 1960s has led to particular interest in, and development of, appropriate logics and type theories, and to many applications of reasoning in specification and verification, that is in proving that programs, distributed systems, hardware devices, or the descriptions of these in some formal system, have certain properties.

In particular the integration of computer algebra and theorem proving has attracted much research interest recently: however thus far it has had little impact on the practice of computer algebra, perhaps because practitioners are not, in general, interested in developing proofs in a formal system. We consider another approach and show how the use of embedded theorem proving, in the form of calls to a black-box theorem prover hidden from the user, can be used to extend the capabilities of computer algebra systems and provide a better solution to a known difficult problem than that currently available.

Symbolic definite integration in the presence of parameters is recognised [15, 41] as a tricky problem where current algorithms are not adequate and computer algebra systems used naively can get even very simple examples wrong, as we shall see below.

We present VSDITLU, a verifiable symbolic definite integral table look-up: a system which matches a query comprising a definite integral with parameters and side conditions, against an entry in a verifiable table and uses a call to a library of lemmas about the reals in the theorem prover PVS [38] to aid in the transformation of the table entry into an answer. We describe both the full model of such a system and our prototype implementation, which is able to obtain correct answers in cases where standard techniques, such as those implemented in Maple and Mathematica, do not. Our system extends the current best available electronic table, Fateman's web-based table [19], by including parametric limits of integration and queries with side conditions, for example:

$$\int_0^1 (x^2 + 2\sin(d)x + 1)^{-1}\, dx \text{ for } |d| < \pi/2$$

The importance of this work lies both in the novelty of verifiable table look up, and, more generally, as an indication of how computer aided theorem proving, particularly embedded verification with library support, can be a valuable tool to support users of mathematics, such as engineers, who want trusted results with minimal user interaction. We argue that rather than expecting computer algebra users to use computer aided theorem proving systems directly, a useful way forward is to provide black-box components which can be incorporated in applications such as VSDITLU.

As far as we are aware the work on symbolic definite integration that we describe here is the first time a theorem prover has been used in developing such a machine look up table. In the next sections we describe the problem of symbolic integration in more detail, develop the necessary background in automated theorem proving and indicate why look-up tables are valuable. Section 4 contains a sketch our implementation and examines the theorem proving issues involved.

*This work is supported by the UK EPSRC: grant GR/L48256.

2 Symbolic definite integration

As an example of the difficulties of symbolic definite integration consider

$$\int_0^c \frac{1}{x-1}\,dx$$

for which, used in naive mode, both Maple 4 and Mathematica 3 return $\mathrm{Ln}[c-1]-i\pi$ (where Ln denotes the principal value of the complex logarithm function) and hence an incorrect complex answer for $c>1$. This is a consequence of misapplying the fundamental theorem of calculus to the indefinite integral $\mathrm{Ln}[x]$ through a pole at $x=1$. Even on an indefinite integrals both systems can be caught out, returning

$$\int x^{(-a-1)/(a+1)}\,dx = \frac{x^{1+(-a-1)/(a+1)}}{1+(-a-1)/(a+1)} = 1/0$$

by failing to simplify $(-a-1)/(a+1)$ and hence matching incorrectly against the generic case $n\neq -1$ of $\int x^n\,dx$.

Problems can arise from the incorrect handling and propagation of pre- and side-conditions, simplifying expressions in elementary functions (for which there is no canonical form or decision procedure) or in combining and matching parameterised expressions under constraints. A more fundamental problem involves the handling of functions over the reals and the blurring of computer algebra and computer analysis. CAS compute indefinite integrals by computing antiderivatives using the Risch algorithm, which can be developed entirely algebraically through the theory of differential rings [5]. However a correct symbolic definite integration procedure needs to work not only algebraically but analytically as well: considering poles, zeros and domains of definition of elementary functions, and here computational techniques are less well-developed. In particular, since continuity is undecidable, any algorithm must work with more tractable stronger pre-conditions: for example using a syntactic decomposition of the function to check for potential poles [17].

This illustrates a more general design issue: there are many examples of processes, like definite symbolic integration using the fundamental theorem of calculus, where a CAS may be able to compute an answer, sometimes correct, on a large class of inputs (any function where Risch returns an indefinite integral), be provably sound on only a subclass of those inputs (where the function is continuous) and be able to check soundness easily on a smaller subclass still (functions with no potential poles). Thus the suppression of pre- and side-conditions is a design decision for ease of use. Some CAS such as **axiom** are cautious: only giving an answer when pre-conditions are satisfied. Mathematica and Maple used in naive mode attempt to return an answer whenever they can and leave to the user the burden of checking correctness.

The Davenport plan [15] describes how to combine symbolic computation and theorem proving to get a better implementation than is presently possible using computer algebra techniques alone. Roughly speaking this analyses the function for possible poles, uses these to decompose the range of integration into components on which the integrand is continuous, applies the Risch algorithm and the Fundamental Theorem of Calculus on each component, then combines and simplifies the answers, all in the presence of parameters. Notice that the presence of parameters imposes an extra complication: when we have identified the potential poles we have to determine in what orders they can lie on the real line to determine the intervals over which our function may be piecewise continuous.

Table-look up is recognised as a useful alternative to the problem, and paper look-up tables have a long history [25, 48], though all the published paper look-up tables contain errors [34]. As well as avoiding the difficulties with computer algebra systems, particularly for non-expert users, they allow answers to be recorded which algorithms such as Risch do not cover, or return in unexpected forms. As has been argued by Fateman [19] machine look-up tables have obvious advantages over paper ones, offering automated pattern matching, ability to handle far more complex table entries and side conditions, and interoperability with other software. In particular web-based tables can be routinely updated with new results, and allow sharing and reuse of entries which may be complicated to obtain and are likely to be useful to other practitioners, as well as providing interesting opportunities for investigating user demand (which is often for nothing more difficult than homework problems). However electronic versions of paper tables are not always satisfactory: the CRC Standard Math Interactive CD [47] still contains errors, and does not offer the facilities one might expect, such as automatic matching and simplification of integrals against user input, or exporting in a standard interchange format such as OpenMath [14], which is not surprising when several of the formulae seem to be stored only as images! There are at least two web-based look-up tables: Fateman's [20] handles symbolic definite integrals with numeric limits of integration and includes all of the CRC entries without parametric limits of integration. The Mathematica table [46] is limited to indefinite integrals and appears to return erroneous results.

Verifiable machine look up tables offer a greater possibility of freedom from error through the use of machine certification of the table entries, and enhanced machine support for transformation to a simplified answer using the side conditions. Thus we are led to consider the problem of a verifiable symbolic definite integral table look-up, which bypasses the difficulty of verifying the integration algorithm or finding the correct definite integrand by verifying particular table entries. A user submits a query with side conditions which is matched against the table system which matches a query comprising a definite integral with parameters and side conditions, against an entry in a verifiable table and uses a call to a black-box for solving inequalities over the reals to aid in the transformation of the table entry into an answer.

3 Computer aided theorem proving

In the early years of computing several logicians experimented with implementing decision and semi-decision (an algorithm which returns "yes" if a condition is satisfied but fails to terminate otherwise) procedures for various fragments of logic, culminating in much work on Robinson's resolution, a semi-decision procedure for first order logic later incorporated into the language Prolog. Most interesting mathematics is neither first order nor semi-decidable, and an alternative approach was the development of proof checkers: a user or another program inputs a proof in a particular logical system and the program checks it. Theorem provers in the LCF tradition automated this notion, using tactics which combined primitive inference steps to build up proofs automatically. de Bruijn's AUTOMATH [16], developed in 1967, which was used by Jutting to check the proofs from Landau's Grundlagen der Analysis was an early system of this kind: the development of these ideas using various logics and type theories has led to widely used systems such as HOL[24] and Isabelle [39]. More recently systems such as PVS [38] have combined various techniques in one platform.

There has been much work in the theorem proving community on the development of formalised mathematics. Thus for example the foundations of real analysis have been developed in several

theorem provers, following textbook accounts of the definition of the reals, continuity, indefinite integration, the fundamental theorem of calculus and so forth: for example AUTOMATH [16] and Mizar [43]. Harrison [26] developed a large portion of real analysis in HOL-light, both major theorems and also setting in place the mechanisms (power-series and so forth) to prove low-level lemmas about elementary functions. The reals are constructed by means of Dedekind cuts. By contrast Dutertre [18] uses an axiomatic approach, extending the built in axiomatisation of the reals, to prove results about the reals in PVS, and again proves both major theorems and more low-level results. Fleuriot [21] has also implemented both classical and non-standard reals in Isabelle.

Roughly speaking it is possible with enough work to mechanise just about any piece of sufficiently developed existing mathematics, filling in the details of whatever outline is provided. The difficulties come in the sheer amount of work involved, in managing the size, scale and efficiency of the operation, organising all the lemmas that are required to aid re-use and in ensuring that the details fit together exactly without falling back on that familiar stand-by "and similarly". While modern systems allow automation through the use of high level tactics it is still the case that such a development is often "graduate level work". Notice too that theorem provers prove theorems rather than make conjectures or do calculations, and a proof of a simple arithmetical statement, even if performed entirely automatically by means of a suitable tactic, may still be lengthy. Harrison and Théry [27] experimented with combining such development with the use of Maple as an oracle to compute, for example, factorisations of polynomials or indefinite integrals which could then by verified by the prover.

However the success of theorem proving in applications like the verification of hardware has come not so much from using them to develop proofs of difficult theorems, but for using them in automatic or semi-automatic mode to develop proofs of large numbers of rather routine low level results for which hand proofs would be easy if tedious and error-prone. Likewise for applications like VSDITLU or the Davenport plan one needs not so much developments of text-book theorems but a succession of proofs that particular fairly straightforward functions are defined, continuous and so on. Even if dealing with an undecidable problem like continuity it is possible to put together a black-box which incorporates a large database of previously proved useful lemmas (properties of elementary functions for example) and tries a variety of approaches automatically

An early objective of theorem proving was proofs of correctness of programs: this has been replaced in recent years with an emphasis on developing specifications, proving they have required correctness properties and developing these into code while retaining correctness. However this does not address the problem of poor behaviour of symbolic definite integration: we do not have a very satisfactory algorithm to verify. A fully verified implementation of a "correct" algorithm would involve a fully verified implementation of the major part of a computer algebra system, as well as several graduate level textbooks such as [5]: this is beyond the scope of what is currently feasible. At least such proofs, only being required once (until the system was upgraded), could be carried out by a theorem proving expert. However on the fly proofs would also be needed each time it was run to verify pre-conditions (for example continuity) on the input.

A significant strand in computational logic has been constructive methods, which extract a program from a proof: for example Théry [42] provided a constructive derivation of the basic version of the Gröbner basis algorithm (i.e. of a version without many of the optimisations commonly in use). As outlined in [29] doing computer algebra this way requires a major change in approach, for example the usual definition of an ideal needs to be replaced by one that gives a test for ideal membership.

The integration of computer algebra and theorem proving has been something of a hot topic recently. On the one hand is work such as [29, 42], whose long term goal is to develop substantial portions of computer algebra inside a theorem prover: to do this properly requires formalising large amounts of mathematics, with a great loss in efficiency if standard algorithms such as factorisation are implemented from first principles. Various hybrid systems have been proposed, where theorem provers call computer algebra systems [28], either using the CAS as an oracle whose results are then then proved in the theorem prover[27], or arranging for the CAS to provide hints or plans towards proofs of its results [32], or to trust the implementation of CAS while the theorem prover does the book-keeping of checking interface definitions and so on [4]. The resulting increase in precision, for example requiring side-conditions to be handled correctly at every stage, may be unwelcome to mathematicians who are used to using computer algebra systems in a fairly informal way. Thus such efforts may extend our trust of computer algebra systems, but at a cost of performance or flexibility.

While there has been an enormous amount of research in computer aided theorem proving and its applications to mathematics and computer algebra its impact on these domains has been less than might have been hoped. As one of us has argued elsewhere [37] this is because much such research fails to address current mathematical practice: such systems are difficult to use, neither practising mathematicians nor users of mathematical software are much interested in producing formal proofs, and the size of the task makes infeasible either proving correct an existing computer algebra system or developing a new correct computer algebra system from a formal specification, even if agreement could ever be reached on what that specification should be.

Perhaps the greatest success of computer aided theorem proving in mathematics has been in efficient techniques for particular subproblems, such as quantifier elimination or resolution, and in the development of ideas such as type theories, unification algorithms or quantifier elimination which can then be incorporated directly in CAS. Examples include the **axiom**[30] type system and quantifier elimination in REDUCE [45] Another interesting development in this area is Theorema [7, 8], which is a project aimed at developing general theorem proving support within Mathematica and which now includes special purpose routines for induction over the natural numbers and lists as well as a few other areas. The project also aims to allow standard algorithms to be developed and proved correct. Another similarly successful application is Clarke's Analytica system [12], which extended Mathematica with a collection of inference rules, including first order natural deduction and induction, implemented using its built in matching, simplification and rewriting algorithms, and was able to prove identities involving finite and infinite sums from Ramanujan's notebooks. Theorem proving technology has also influenced applications such as Dalmas' Mathematical Formula Database [13], a deductive database of mathematical formulae coded in ML (re-implementing logic programming techniques) which uses sophisticated unification algorithms to attempt to answer queries such as "Is X true?" or "Simplify X", where X is an expression over the reals involving elementary functions, integrals, derivatives and so on.

For our purposes in implementing the Davenport plan or the VSDITLU we need to use a theorem prover whose base logic is compatible with that of classical mathematics, viz classical first or higher order logic with induction, and we need an implementation of real numbers, elementary functions and the basics of real analysis within that prover. Since we want our system to call the prover with minimal user interaction to prove simple lemmas over the reals we want a high level of automated proof available for this

theory, although we are initially uninterested in whether this is a consequence of general automation or theory-specific automation.

We next outline the VSDITLU, before returning to our discussion of theorem proving.

4 The VSDITLU

We describe the principle of the VSDITLU, and our prototype implementation, and discuss the theorem proving tasks it generates.

We are considering expressions of the form

$$\int_b^c f(x, p_1, \ldots, p_n)dx$$

where x is a real variable, $b, c, p_1, \ldots, p_n$ are real [1] parameters, and f is a function over the reals. The VSDITLU comprises verifiable table entries of the form

$$\int_b^c f(x, p_1, \ldots, p_n)dx : K, C \tag{1}$$

where K is a sequence of pairs of the form $\langle A, R\rangle$. Informally such a pair denotes that, under the constraints or side conditions R,

$$\int_b^c f(x, p_1, \ldots, p_n)dx = A,$$

while C records information to assist in verifying the table entry. More precisely A is a real function, or "unknown" or "undefined", R is a boolean combination of equalities and pure inequalities over $\{b, c, p_1, \ldots, p_n\}$ and C is a certificate, a set of assertions for use in validating the entry. Formally the table entry asserts that for all values of the parameters b, c, p_i, and for each $\langle A, R\rangle$ in K we have

$$(C \wedge R) \Longrightarrow \int_b^c f(x, p_1, \ldots, p_n)dx = A.$$

A table entry is said to be complete if it covers all possible values of the parameters, that is to say $\{\overline{R} \mid \langle A, R\rangle \in K\}$ partitions and covers the parameter space, where $\overline{R}$ denotes the solutions of R. Notice that even complete table entries need not be unique, as there may be many different ways of representing the same real function. We discuss the verification of the table entries below.

To use the table the user submits a query

$$\left(\int_{b'}^{c'} g(x, p'_1, \ldots, p'_m)\ dx, Q\right),$$

where $b', c', p'_1, \ldots, p'_m$ are real parameters and Q is a boolean combination of equalities and pure inequalities over $\{b', c', p'_1, \ldots, p'_m\}$. The integral is matched automatically against the integrand of one or more table entries of the form (1) to obtain a match, or more generally a set of matches, Θ. We discuss the matching in more detail below.

[1] In principle we could include additional type constraints such as integer or rational.

Having obtained the matchings, Θ and Q are used to return a table of answers

$$\int_{b'}^{c'} g(x, p'_1, \ldots, p'_n)\ dx : L$$

where L is a set of pairs of the form $< A', R' >$, for A' a real expression and R' a set of constraints. This denotes that for all values of the parameters b', c', p'_i, and for each $\langle A', R'\rangle$ in L we have

$$(R' \wedge Q) \Longrightarrow \int_{b'}^{c'} g(x, p'_1, \ldots, p'_n)\ dx = A'.$$

To solve for L we note first that for any $\langle A, R\rangle \in K$ and $\theta \in \Theta$ we have, for all values of the parameters b', c', p'_i that

$$R\theta \Longrightarrow \int_{b'}^{c'} g(x, p'_1, \ldots, p'_m)\ dx = A\theta,$$

and hence

$$R\theta \wedge Q \Longrightarrow \int_{b'}^{c'} g(x, p'_1, \ldots, p'_m)\ dx = A\theta$$

and so if K is complete we may take

$$L = \{\langle A\theta, R\theta \wedge Q\rangle \mid \langle A, R\rangle \in K, \theta \in \Theta\}.$$

However if for some θ and R the set of constraints $R\theta \wedge Q$ has no solutions in b', c', p'_i, that is if

$$\neg\exists\, b', c', p'_i\ .\ R\theta \wedge Q, \tag{2}$$

then $\langle A\theta, R\theta \wedge Q\rangle$ contributes no extra solutions to L, and so we may assume

$$L = \{\langle A\theta, R\theta \wedge Q\rangle | \langle A, R\rangle \in K, \theta \in \Theta, (\exists\, b', c', p'_i\ .\ R\theta \wedge Q)\}.$$

Thus each $\langle A, R\rangle \in K$ gives rise to a side condition (2), and if the side condition is true $\langle A, R\rangle$ does not contribute to L. We discuss the verification of these side conditions below.

Figure 1 presents a typical VSDITLU entry, omitting the certificate C.

So if a user enters the query:

$$\left(\int_l^m \frac{1}{cos(d) + 2x}\ dx, (m > 3) \wedge (l > 3)\right)$$

the VSDITLU should match this with unifier

$$\phi = \{cos(d) \leftarrow p, 2 \leftarrow q, l \leftarrow b, m \leftarrow c\},$$

giving rise to five side-conditions:

$$\neg\exists\, l, m, d.[b = c]\phi \wedge l > 3 \wedge m > 3,$$

$$\neg\exists\, l, m, d.[(q \neq 0) \wedge (b \neq c) \wedge ((b = -\frac{p}{q}) \vee (c = -\frac{p}{q}))]\phi \wedge l > 3 \wedge m > 3,$$

$$\neg\exists\, l, m, d.[(q \neq 0) \wedge (b \neq c) \wedge (b \neq -\frac{p}{q}) \wedge (c \neq -\frac{p}{q})]\phi \wedge l > 3 \wedge m > 3,$$

$$\neg\exists\, l, m, d.[(b \neq c) \wedge (p \neq 0) \wedge (q = 0)]\phi \wedge l > 3 \wedge m > 3,$$

$$\neg\exists\, l, m, d.[(b \neq c) \wedge (p = 0) \wedge (q = 0)]\phi \wedge l > 3 \wedge m > 3.$$

$$\int_b^c \frac{1}{p+qx}\,dx =$$

Answer	*Constraints*
0	$(b=c)$
undefined	$(q\neq 0)\wedge(b\neq c)\wedge$ $((b=-\frac{p}{q})\vee(c=-\frac{p}{q}))$
$\frac{\mathrm{Ln}\lvert qc+p\rvert-\mathrm{Ln}\lvert qb+p\rvert}{q}$	$(q\neq 0)\wedge(b\neq c)\wedge$ $(b\neq -\frac{p}{q})\wedge(c\neq -\frac{p}{q})$
$\frac{c-b}{p}$	$(b\neq c)\wedge(p\neq 0)\wedge(q=0)$
undefined	$(b\neq c)\wedge(p=0)\wedge(q=0)$

Figure 1: A Typical VSDITLU Entry

Of these the second, fourth and fifth are true (a formal proof would use the fact that $\forall x.\ -1\leq\cos(x)\leq 1$) and so we obtain the answer

$$\int_l^m \frac{1}{cos(d)+2x}\,dx =$$

Answer	*Constraints*
0	$(l=m>3)$
$\log\frac{\lvert 2m+cos(d)\rvert}{\lvert 2l+cos(d)\rvert}$	$(l\neq m)\wedge(l>3)\wedge(m>3)$

Notice that for concision we have simplified the constraints remaining in the answer. There is no canonical method for doing this, though sometimes obvious redundancies and subsumption may be eliminated.

5 Theorem proving issues

We now discuss our theorem proving tasks in more detail. These can be characterised as

- validating the table entries
- matching a query against the table
- verifying the side conditions to return a result

Validating the table entries

To add a new entry to the table it is necessary to provide K and C and to verify that the entry is correct. There are two parts to this verification: showing that the entry (1) is correct and showing that the entry is complete.

We described above what would be involved in computing and verifying an integral from scratch via the Risch algorithm. What we propose here is more straightforward: we assume that the computation has already been done, possibly by ad hoc means or by calling upon an existing table, and all that is required is to validate the result. The certificate C allows us to provide auxiliary lemmas to assist in this. Nonetheless it is still unlikely that except in the very simplest of cases this verification could be carried out automatically unless each certificate included a proof outline drawn up by a domain expert: the proof needs to call on a rich lemma database of facts about continuity, singularities, elementary functions and so forth.

We have not yet worked out a format for C suitable for a production version of the VSDITLU. It seems likely that proof planning [32] will be useful here: the certificate might comprise a full proof plan or a standard template with information about poles and so forth for use with a pre-prepared proof plan.

The second part of this verification involves showing completeness by showing that $\{\overline{R}\mid\langle A,R\rangle\in K\}$ partitions the parameter space, where $\overline{R}$ denotes the set of solutions of R. We may reduce the theorem proving task by requiring only that $\{\overline{R}\mid\langle A,R\rangle\in K\}$ covers the parameter space, in which case we may have redundancy in our table entry, but it will still cover all cases. We have not yet addressed the problem of partial subsumption or overlap between different entries in the table: this comes back again to problems of representation.

Matching

Matching and unification algorithms are a fundamental part of many symbolic computation systems and theorem provers, and have received considerable practical and theoretical attention: see for example [3] for a survey. The most general form of matching here is undecidable: we are working over the reals and so for example $x+2$ needs to match $b+x+3$, $x-1/b$, $x+b^2$ and $x+1/b^2$ but not $x-b^2$ or $x-1/b^2$. The best we can hope for is a suite of methods sufficient to cover a wide range of cases: it is common also in computer algebra systems to make the problem more tractable by pre-processing functions to a standardised form, for example $x+a^2$ is represented as $x+c$ with the side condition $a^2=c$. In addition certain forms, such as sums, tend not to occur in integral tables as it is assumed the user has pre-processed a query such as $\int(f+g)\,dx$ to separate queries $\int f\,dx$, $\int g\,dx$. Note also that, while a VSDITLU query may match several entries, it is sufficient for our purposes to find a match against one complete table entry to get the required answer.

While there is a rich literature on matching and unification, as far as we know there is no existing implementation that is entirely adequate for our purposes. In his look-up table Fateman [19] uses pre-processing and stores the integrands in a particular kind of discrimination tree: matching is performed by a succession of approximate matches. Dalmas [13], in his work on a deductive database of mathematical formulae uses a similar data-structure together with a conditional AC-unification algorithm. Both systems appear to be correct but not complete, that is there are matches which they fail to find.

The pattern matching algorithm implemented for our prototype system is very simple and in fact takes no account of units and zeros for the AC operators $+$ and $\times$, although it does perform full AC matching over these operators. Expressions are limited to syntactically to terms in combinations of integers, π and e with the function symbols $\sin,\cos,\tan,\sinh,\cosh,\tanh$ and their inverses, together with modulus ($||$), Ln and rational exponentiation.

Proving the side conditions

The side conditions have the form

$$\neg\exists\, b, c, p_i\ .\ H$$

where H is a boolean combination of equalities and strict inequalities involving real functions over $\{b, c, p_1, \ldots, p_n\}$.

For polynomial functions the problem may be addressed using quantifier elimination [6] which solves the complementary problem

$$\exists\, b, c, p_i \;.\; H.$$

Extending this to other functions such as $\exp$ is currently an active research area, and in any case these methods are intractable for all but the smallest examples.

Thus we chose to pass the side conditions to the theorem prover PVS for automatic verification. PVS [38], the Prototype Verification System, is a theorem prover implementing classical higher order logic. It has a typed specification language to which new types may be added, including predicate subtypes, types formed from existing types by specifying a predicate which members of the subtype must satisfy (i.e. $\{x : T|P(x)\}$). PVS has an existing type of *Real* numbers, defined axiomatically as an uninterpreted subtype of *number*: rationals, integers and natural numbers are defined as subtypes of the reals in a similar manner. In addition, PVS contains definitions and theorems for built-in data structures such as sequences. Dutertre [18] developed the basics of real analysis: convergence, continuity, differentiation and so on in PVS.

To prove our side conditions automatically the prover needs to be able to call upon a library of standard facts about the elementary functions, for example $\forall x. -1 \leq \cos(x) \leq 1$. It is possible to add elementary functions as uninterpreted function symbols with type $\mathbb{R} \Rightarrow \mathbb{R}$, and then to add the facts we need as axioms, but this is not very satisfactory and prone to errors which would lead to inconsistency. Instead we are extending Dutertre's library by defining elementary functions in terms of series and them proving a large collection of the necessary facts as lemmas, following the work of Harrison [26] in HOL-Light.

A major point in favour of PVS for use in our prototype was the inclusion of a very powerful generic automated proof tactic *grind* which applies a brute force search in an attempt to prove the required results. Grind in turn is calling built-in PVS procedures to handle Boolean combinations and inequalities: if for example the inequalities happen to be linear a further efficient built-in decision procedure can be called.

6 Our implementation

Our implementation consists of a front end comprising the table entries and a matching algorithm: around 2000 lines of Allegro Common Lisp. At present table entries and calls must be in a fairly strict standard form and we do not do any additional simplifications or redundancy checks: in principle our front end could be interfaced to a computer algebra system such as **axiom** for pre- and post-processing so as to handle a wider variety of inputs. The standard form aids the matching which is currently a basic form of AC pattern matching which makes no attempt to account for the units $(1, 0)$ of the AC operators $(+, *)$. For a full description see [1]. Some of the table entries have been verified though we have not yet developed the notion of certificate very precisely.

PVS is called through emacs to prove the side conditions as described above and return an answer. Our table currently contains six entries and we have been able to evaluate correctly around 60 examples from a test suite of symbolic definite integrals that CAS running in a naive mode were unable to evaluate or got wrong, mainly involving rather simple variants on elementary functions with parameters across poles. Our implementation got no answers wrong, but it did sometimes fail to return an answer because our matching algorithm was not powerful enough. On one occasion *grind* sent PVS into an apparently infinite loop and it was unable to identify an obvious counter-example to the non-existence of values for the parameters in a particular theorem.

The integrals in the table are:

$$\int_b^c \frac{p}{x^2+a}\,dx\;(*) \qquad \int_b^c \frac{1}{x^2+a}\,dx\;(*)$$

$$\int_a^b x\tan^{-1}(\frac{1}{x})\,dx \qquad \int_{l_1}^{l_2} \frac{p}{bx+a}\,dx\;(*)$$

$$\int_{l_1}^{l_2} \frac{y}{by+a}\,dy \qquad \int_{l_1}^{l_2} \frac{1}{x+a}\,dx\;(*)$$

The integrals marked $(*)$ have a complete set of answers. Each of these may contain poles in the range of the integration and CAS give answers which are incorrect for some values of the parameters, even when using the "assumes' facility of Maple V to include constraints on the parameters. No CAS that we tested (Maple V, Mathematica 3, **axiom** and Matlab) was able to consistently produce full correct answers to these integrals: in fact, all these CAS produce incorrect answers to some of them.

Development time was very short: 3-4 person months. Our choice of PVS was a fairly pragmatic one, based on what system had the best real library available in August 1998 and the effectiveness of the *grind* command: more recent work in HOL makes it also a suitable candidate system.

7 Further Work

Our existing system is fairly limited but has, we believe, shown proof-of-concept of verifiable symbolic definite integral table look-up with embedded verification support. We envisage the possibility of a production quality VSDITLU supported as a component of an environment such as NAG/WMI's PSE, whose scope includes the entries of paper tables such as CRC [48].

Obvious areas for further work include developing our ideas of how to verify the table entries, improving the matching algorithms and search techniques, refining our techniques for proving the side conditions and including calls to a computer algebra systems for pre- and post-processing of the results. In particular the development of more effective matching techniques for real elementary functions, extending those used by Fateman and Dalmas, would have widespread application. Likewise there would be other uses for a more effective black-box for handling inequalities over elementary functions, or for a richer verified library of lemmas about elementary functions over the reals which would support it.

As mentioned above, our choice of PVS was a pragmatic one based on what was available in August 1998. In addition, the requirements for a prototype, where one is attempting proof of concept, are different to those for a full production system. Much more time could be spent developing special purpose decision procedures within HOL, Isabelle or PVS for a full production VSDITLU rather than relying on existing general purpose automation.

The VSDITLU is part of a wider project to implement the Davenport plan [15]: this requires further black-box verification of matters such as continuity or behaviour at potential poles.

We have argued that much research on the application of theorem proving to computer algebra, while important for the development of computational logic, is not yet at a stage when it is

helpful to the computer algebrist. We illustrate this by considering symbolic definite integration. Implementing this inside a theorem prover would be extremely hard, would probably need to calls to a computer algebra system for matters such as factorisation and would give greater security at a cost of performance and flexibility. Similar problems would arise in verifying an algorithm for symbolic definite integration, such as the Davenport plan.

However embedded theorem proving, in the role of black-boxes for matters such as continuity or solving inequalities, can extend the scope and increase the reliability of processes such as the VS-DITLU or the Davenport plan while shielding users from the technicalities of computer aided theorem proving. As yet their design has received little attention in the theorem proving community: hence our implementation is rather crude and relies on nothing more than a powerful brute force search algorithm and a large pre-prepared lemma database. Yet the success of our implementation suggests it would be profitable to give then further attention.

Acknowledgements

We acknowledge the support of the UK EPRSC under grant number GR/L48256 and of NAG Ltd, and we thank Mike Dewar from NAG for his interest and suggestions, James Davenport and Richard Fateman for advice on computer algebra, and Bruno Dutertre for allowing us to extend his code for the reals in PVS.

References

[1] ADAMS, A. A., GOTTLIEBSEN, H., LINTON, S. A., AND MARTIN, U. A Verifiable Symbolic Definite Integral Table Look-Up. Tech. Rep. CS/99/3, University of St Andrews, 1999.

[2] ALUR, R., AND HENZINGER, T. A., Eds. *Computer aided verification: 8th international conference* (1996), Springer-Verlag LNCS 1102.

[3] BAADER, F., AND SIEKMANN, J. H. Unification theory. In Gabbay et al. [22], pp. 41–125.

[4] BALLARIN, C., HOMANN, K., AND CALMET, J. Theorems and Algorithms: An Interface between Isabelle and Maple. In Levelt [36], pp. 150–157.

[5] BRONSTEIN, M. *Symbolic integration (I)*. Springer-Verlag, Berlin, 1997. Transcendental functions.

[6] BROWN, C. W. Simplification of truth-invariant cylindrical algebraic decompositions. In Gloor [23], pp. 295–301.

[7] BUCHBERGER, B. Symbolic computation: computer algebra and logic. *Appl. Log. Ser. 3* (1996), 193–219.

[8] BUCHBERGER, B., JEBELEAN, T., KRIFTNER, F., MARIN, M., TOMUŢA, E., AND VĂSARU, D. A Survey of the Theorema Project. In Küchlin [35], pp. 384–391.

[9] BUNDY, A., Ed. *CADE-12: 12th International Conference on Automated Deduction: Proceedings* (1994), Springer-Verlag LNAI 814.

[10] CALMET, J., AND CAMPBELL, J. A., Eds. *Integrating Symbolic Mathematical Computation and Artificial Intelligence* (1994), Springer-Verlag LNCS 958.

[11] CALMET, J., AND LIMONGELLI, C., Eds. *Design and Implementation of Symbolic Computation Systems, International Symposium, DISCO '96* (1996), Springer-Verlag LNCS 1128.

[12] CLARKE, E., AND ZHAO, X. Combining symbolic computation and theorem proving: Some problems of Ramanujan. In Bundy [9], pp. 758–763.

[13] DALMAS, S., GAËTANO, M., AND HUCHET, C. A Deductive Database for Mathematical Formulas. In Calmet and Limongelli [11].

[14] DALMAS, S., GAËTANO, M., AND WATT, S. An OpenMath 1.0 Implementation. In Küchlin [35], pp. 241–248.

[15] DAVENPORT, J. H. Really Strong Integration Algorithms. In preparation.

[16] DE BRUIJN, N. G. A Survey of the Project AUTOMATH. In Seldin and Hindley [40], pp. 579–606.

[17] DUPÉE, B. Using Computer Algebra to Find Singularities of Elementary Real Functions. Available from the author, bjd@maths.bath.ac.uk, 1998.

[18] DUTERTRE, B. Elements of Mathematical Analysis in PVS. In von Wright et al. [44].

[19] EINWOHNER, T., AND FATEMAN, R. J. Searching techniques for Integral Tables. In Levelt [36], pp. 133–139.

[20] FATEMAN, R. J., AND EINWOHNER, T. TILU Table of Integrals Look Up. Web Service. http://http.cs.berkeley.edu/∼fateman/htest.html.

[21] FLEURIOT, J., AND PAULSON, L. A combination of nonstandard analysis and geometry theorem proving with application to Newton's Principia. In Kirchner and Kirchner [33], pp. 3–16.

[22] GABBAY, D. M., HOGGER, C. J., AND ROBINSON, J. A., Eds. *Handbook of logic in artificial intelligence and logic programming. Vol. 2*. OUP, 1994.

[23] GLOOR, O., Ed. *Proceedings of the 1998 International Symposium on Symbolic and Algebraic Computation* (1998), ACM Press.

[24] GORDON, M. J. C., AND MELHAM, T. F., Eds. *Introduction to HOL*. CUP, 1993.

[25] GROËBNER, W., AND HOFREITER, N. *Integraltafel*. Springer-Verlag, Vienna, 1961.

[26] HARRISON, J. *Theorem Proving with the Real Numbers*. Springer-Verlag, 1998.

[27] HARRISON, J., AND THÉRY, L. Extending the HOL theorem prover with a computer algebra system to reason about the reals. In Joyce and Seger [31], pp. 174–185.

[28] HOMANN, K., AND CALMET, J. Combining theorem proving and symbolic mathematical computing. In Calmet and Campbell [10], pp. 18–29.

[29] JACKSON, P. J. Exploring Abstract Algebra in Constructive Type Theory. In Bundy [9].

[30] JENKS, R. D., AND SUTOR, R. S. *AXIOM*. Springer-Verlag, 1992.

[31] JOYCE, J. J., AND SEGER, C.-J. H., Eds. *Higher order logic theorem proving and its applications* (Berlin, 1994), Springer-Verlag LNCS 780.

[32] KERBER, M., KOHLHASE, M., AND SORGE, V. Integrating Computer Algebra with Proof Planning. In Calmet and Limongelli [11].

[33] KIRCHNER, C., AND KIRCHNER, H., Eds. *CADE-15: 15th International Conference on Automated Deduction: Proceedings* (1998), Springer-Verlag LNAI 1421.

[34] KLERER, M., AND GROSSMAN, F. Error Rates in Tables of Indefinite Integrals. *Journal of the Industrial Mathematics Society 18* (1968), 31–62.

[35] KÜCHLIN, W. W., Ed. *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation* (1997), ACM Press.

[36] LEVELT, A. H. M., Ed. *Proceedings of the 6th International Symposium on Symbolic and Algebraic Computation, ISSAC '95* (1995), Springer-Verlag LNCS 1004.

[37] MARTIN, U. Computers, Reasoning and Mathematical Practice. In Computational Logic, NATO Adv. Sci. Inst. Ser. F Comput. Systems Sci., 1998.

[38] OWRE, S., RAJAH, S., RUSHBY, J. M., AND SHANKAR, N. PVS: combining specification, proof checking, and model checking. In Alur and Henzinger [2], pp. 411–414.

[39] PAULSON, L. C. *Isabelle: A Generic Theorem Prover*. Springer-Verlag LNCS 828, 1994.

[40] SELDIN, J. P., AND HINDLEY, J. R., Eds. *To H.B. Curry: essays on combinatory logic, lambda calculus and formalism*. Academic Press, 1980.

[41] STOUTEMYER, D. Crimes and misdemeanours in the computer algebra trade. *Notices of the AMS 38* (1991), 779–785.

[42] THÉRY, L. A Certified Version of Buchberger's Algorithm. In Kirchner and Kirchner [33], pp. 349–364.

[43] TRYBULEC, A. The Mizar-QC 6000 logic information language. *ALCC Bulletin 6* (1978), 136–140.

[44] VON WRIGHT, J., GRUNDY, J., AND HARRISON, J., Eds. *Theorem Proving in Higher Order Logics: 9th International Conference* (1996), Springer-Verlag LNCS 1125.

[45] WEISPFENNING, V. Simulation and optimization by quantifier elimination. *Journal of Symbolic Computation 24: Applications of quantifier elimination* (1997), 189–208.

[46] WOLFRAM RESEARCH INC. The integrator: the power to do integrals as the world has never seen before, http://www.integrals.com.

[47] ZWILLINGER, D. Standard Math Interactive. CD-ROM, 1998.

[48] ZWILLINGER, D., KRANTZ, S. G., AND ROSEN, K. H., Eds. *CRC standard mathematical tables and formulae*, 30th ed. CRC Press, Boca Raton, FL, 1996.