Verified Interactive Computation of Definite Integrals

Runqing Xu[1,2], Liming Li[1], Bohua Zhan[1,2(✉)]

[1]SKLCS, Institute of Software, Chinese Academy of Sciences, Beijing, China
[2]University of Chinese Academy of Sciences, Beijing, China
{xurq, lilm, bzhan}@ios.ac.cn

Abstract. Symbolic computation is involved in many areas of mathematics, as well as in analysis of physical systems in science and engineering. Computer algebra systems present an easy-to-use interface for performing these calculations, but do not provide strong guarantees of correctness. In contrast, interactive theorem proving provides much stronger guarantees of correctness, but requires more time and expertise. In this paper, we propose a general framework for combining these two methods, and demonstrate it using computation of definite integrals. It allows the user to carry out step-by-step computations in a familiar user interface, while also verifying the computation by translating it to proofs in higher-order logic. The system consists of an intermediate language for recording computations, proof automation for simplification and inequality checking, and heuristic integration methods. A prototype is implemented in Python based on HolPy, and tested on a large collection of examples at the undergraduate level.

Keywords: Symbolic integration, User interface, Proof automation

1 Introduction

Symbolic computation is an important tool in mathematics, science, and engineering. It forms a key part of many mathematical proofs. On the engineering side, justifications for the design of signal processing and control systems contain extensive symbolic computations [6,33], involving derivatives and integrals, Laplace and Fourier transforms, and various special functions.

Typically, these computations can be performed using computer algebra systems such as Mathematica, Maple, and Maxima. Given the complexity of the task, it is not surprising that even the best of these systems are liable to errors. One famous example is $\int_{-1}^{1} \sqrt{x^2}\, dx$, which an early version of Maple evaluates to zero [23] (the error has been fixed in the more recent versions). Bugs in Mathematica have also been observed by mathematicians [15], including evaluation of determinants of matrices with large integer entries, and several evaluations of integrals (also fixed in the most recent version). While some errors are simply implementation mistakes, more systematic errors in symbolic computation

may arise due to neglect of checking side conditions, involving concepts such as well-definedness of expressions, singularities, convergence, and so on. While individual bugs can be reported and fixed, completely eliminating the possibility of error would require a more systematic approach.

Formalization of mathematics in interactive theorem provers promises to eventually achieve this goal. There is already a lot of work on formalization of analysis and linear algebra in interactive theorem provers, as well as verified computations based on the formalized theories. They provide much stronger guarantees of correctness, and also allow users to specify more detailed steps, enabling computations that are too difficult to be found automatically by computer algebra systems. However, a major disadvantage (for now) is that interactive theorem proving requires a great deal of time and expertise on the part of the user, making it difficult to apply on a much larger scale.

It is therefore natural to try to combine the advantages of computer algebra systems with theorem proving. There have already been many works in this direction. A common approach, proposed by Harrison and Théry [20,23], is to invoke a computer algebra system for computations that are difficult to perform, but whose results can be verified more easily. This greatly extends the capability of proof assistants for tasks such as factorization [23], linear arithmetic [28], etc. However, to use such a system, the user still needs expertise in the use of proof assistants, and the range of applicability is limited by the simple proof automation that is available for checking results.

In this paper, we propose a more general framework for verified symbolic computation in theorem provers, and demonstrate it using computation of definite integrals. The resulting system allows users to perform calculations of definite integrals step-by-step, in a user interface similar to that of a computer algebra system, but with the computations verified by automatic translation to proofs in higher-order logic. We choose definite integration for demonstration purposes, due to the great variety of techniques that can be used, but we intend the idea to be applicable to other kinds of symbolic computations.

The framework consists of several components. At the top, a graphical user interface displays the current computation and allows user actions. The user interface produces computations in a standard format. Next, proof automation is used to reconstruct from the computation a proof in higher-order logic. Finally, the proof depends on theorems in mathematics, e.g. (in the case of definite integration) those concerning continuity, derivatives, and integrals.

We implement a prototype based on HolPy, a new interactive theorem prover written in Python [49]. The SymPy package for symbolic computation in Python is used at various places for untrusted computations. The user interface is written in JavaScript as a web application, using Python as backend for convenient invocation of HolPy and SymPy libraries. The underlying theorems in analysis are mostly translated to HolPy from HOL Light (with some modifications). Their proofs have not been fully formalized in HolPy, hence the statements of these theorems still need to be trusted.

We now give an outline for the rest of this paper. Section 2 presents the overall framework. Section 3 describes the intermediate format for recording computations of definite integrals. In Section 4 and 5, we describe respectively the user interface and the proof reconstruction process. In Section 6 we present an evaluation of the system, along with some interesting examples. Finally, we conclude in Section 7 with discussion of possible future work.

Related work. There is a huge body of work on formal verification of continuous and hybrid systems, based on reachability checking [4], computation of invariants [36,41], deductive methods [34,35,47], and so on. In particular, KeYmaera X [18] provides a user interface for verifying hybrid systems using differential dynamic logic, with automatic generation of proofs checkable in Isabelle [9]. Most of this work focuses on automatic verification and/or logical formalisms. Our work can be seen as complementary, focusing on verifying symbolic reasoning about mathematical concepts such as special functions and integration, which can also form a part of the justification of control systems.

Harrison and Théry proposed the "skeptical" approach for combining theorem provers with computer algebra systems [20,23]. Some common applications include factorization of polynomials, which is further applied to verify antiderivatives involving sine and cosine [23]. More recently, this technique is used by Chyzak et al. to formalize the proof of irrationality of $\zeta(3)$ [14], and by Harrison to verify proofs of hypergeometric sums found using the WZ method [22]. Similar approaches are implemented in Isabelle [8], PVS [3] and Lean [28]. Compared to this work, we present more complex proof automation for reconstructing proofs, as well as a user interface for allowing users to perform multi-step computations in a more familiar setting. Other user interfaces for proof assistants with support for displaying mathematical computations include Theorema [11] and jsCoq [5].

The theory of integration has been formalized in every major proof assistant [12,24,31,40,43]. Recently, more advanced concepts that are important in science and engineering have been formalized, including the work by Hasan et al. on Fourier and Laplace transforms [37,38,46], and Immler et al. on ordinary differential equations [25,26]. Work has also been done on formalizing advanced concepts in linear algebra [29], with applications in analyzing mechanical systems [13,44]. Of course, formalized symbolic computation can be applied in many other domains. For example, Selsam et al. [42] verified in Lean the correctness of stochastic backpropagation, an important algorithm in deep learning.

Slagle initiated the study of automatic integration with a heuristic method [45]. Later research focused more on methods that are complete for certain types of integrands, such as Risch's algorithm [19]. More recently, Rubi (rule-based integration) has been demonstrated to be a powerful technique [39]. However, none of these work focuses on formal verification. A verified computation of asymptotics for real-valued functions is implemented by Eberl [16]. Verified *numerical* computation of definite integrals is implemented by Mahboubi et al. [30].

Acknowledgements. This work was partially supported by the National Natural Science Foundation of China under Grant Nos. 62002351, 62032024, and the

Chinese Academy of Sciences Pioneer 100 Talents Program under Grant No. Y9RC585036.

2 Overall Architecture

In this section, we describe the overall architecture of the system, leaving descriptions of its components to the following sections. We focus on definite integrals of continuous functions in one variable over closed intervals. In particular, we consider expressions given by the following syntax:

$$e := v \mid c \mid e_1 \ op \ e_2 \mid f(e) \mid \mathsf{Deriv}(e, v) \mid \mathsf{Integral}(e, v, a, b)$$

Here v is a variable; c is a constant (either a rational number or π); op is an arithmetic operation ($+, -, \times, \div$ and exponentiation); f is a special function (such as logarithms, exponentials, or trigonometric functions); $\mathsf{Deriv}(e, v)$ denotes the derivative of e with respect to variable v; $\mathsf{Integral}(e, v, a, b)$ denotes the definite integral of e with respect to variable v over the interval $[a, b]$. In the rest of this paper, we will use both concrete syntax and LaTeX form of expressions. We use *locations* to point to particular subexpressions. A location is given by a sequence of natural numbers (written in the form $n_1.n_2 \ldots n_k$, with each n_i starting from zero), specifying the path to a subtree in the abstract syntax tree of an expression. For example, in the expression

$$1 + \mathsf{Integral}(1 + \sin^3(x), x, 0, 1)$$

the location of $\sin^3(x)$ is given by 1.0.1.

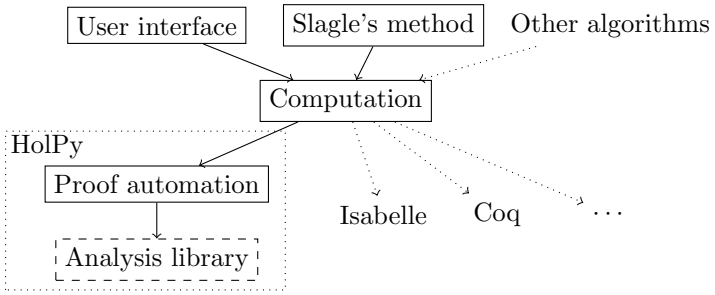
A computation is represented as a list of steps, with each step specifying a rewriting of the current expression. Each step should provide sufficient information so that both checking its correctness and proof generation can be performed relatively easily. A computation begins with the integral to be evaluated, and ends with an expression in simplified closed form. Each step contains the name of the rule used, the location in the expression at which it is applied, and the expected result of applying the step. A step may contain additional parameters and certificates needed for verification. Rules of integration include substitution, integration by parts, use of a trigonometric identity, and so on (described in detail in Section 3). For example, integration by parts takes as parameters two expressions u and v, such that $f \cdot dx = u \cdot dv$ where f is the integrand of the integral at the given location.

A graphical user interface allows the user to specify a computation in ways similar to using a computer algebra system. The user interface displays the computation in LaTeX or in text form. At each step, the user selects part of the current expression to focus on, then selects an action from the menu. Depending on the selected action, the user may need to enter some of the parameters, while the other parameters are automatically inferred by the system. After checking validity of inputs, the user interface computes the result of the action. A package for symbolic computation may be invoked at this step.

There are many side conditions that need to hold in order for a computation step to be correct, some of which may not be caught at the user interface. Translation of the computation to proofs in higher-order logic greatly increases our confidence in the computation and can point out potential errors. In this work, we translate the computation to higher-order logic proofs in HolPy. One main difficulty is implementing sufficiently powerful proof automation for simplification of expressions, inequality checking, and other side conditions. We demonstrate that the API for proof automation in HolPy is sufficiently powerful for this purpose. However, note the representation of a computation is independent from any particular proof assistant, so additional proof translation may be implemented for other proof assistants.

Finally, various algorithms for integration (such as Slagle's method [45]) may be implemented to perform several steps of computation at once. We implemented Slagle's method and have it as one of the options at the user interface.

The overall framework is shown in the following diagram.



Here solid boxes and arrows indicate parts that are implemented for this paper. The analysis library is only partially formalized. Dotted arrows indicate possible future extensions.

This layered design can be viewed as a separation of concerns. At the top, the user only need to think about how to evaluate an integral in general mathematical terms. The implementation of integration algorithms only involves computer algebra. Proof automation involves algorithms for constructing proofs in the underlying logic. Finally, building a library in analysis involves working with a proof assistant. All these are put together to enable verification of potentially difficult symbolic integration by producing proofs in higher-order logic or other logical formalisms. In the following three sections, we describe the top three layers of the system in more detail.

3 Integration Rules

Rules of integration define the language for recording computations. Each rule may take additional parameters (as described below), as well as a location parameter specifying the subexpression the rule is applied on.

3.1 Simplification

The rule **Simplification** rewrites an expression to an equivalent simpler form. The details of simplification depends on the implementation. Here we only specify in broad terms what is and is not simplified. These choices are made mainly considering the ease of performing simplifications, and having a clearly defined "simplified form". We do expand products of polynomials and combine terms (e.g. from $(x+1)(x-1)$ to x^2-1). We do not reduce quotients of polynomials (e.g. from $(x^3+1)/(x^2+1)$ to $x-(x-1)/(x^2+1)$, and from $2/(x^2-1)$ to $1/(x-1)-1/(x+1)$). We do not automatically expand powers (e.g. $(x+1)^5$). We do simplify values of trigonometric functions (e.g. from $\sin(\frac{\pi}{4})$ to $\sqrt{2}/2$, and from $\sin(\frac{\pi}{2}-x)$ to $\cos x$), but do not use other trigonometric identities. We do evaluate derivatives and apply a fixed list of basic integrals, including linearity, powers, sine, cosine, exponential, and derivatives of trigonometric functions.

One complication is that certain rewrite rules contain side conditions. For example, it is only possible to simplify $\sqrt{xy}$ to $\sqrt{x}\cdot\sqrt{y}$ when both x and y are nonnegative. Likewise $(x^2)^{\frac{1}{2}}$ can be simplified to $x^{2\cdot\frac{1}{2}}=x$ only if x is nonnegative (otherwise the mistake mentioned in the introduction would result). When simplifying an integrand of an integral in x, we assume that x is within the open domain of integration, and perform simplification only if it is allowed by this assumption.

3.2 Trigonometric Identities

Application of trigonometric identities can be very tricky. It is often necessary to use trigonometric identities to rewrite an expression to a more complex form, in order to prepare for a substitution or integration by parts.

We use the classification of trigonometric identities by Fu et al. [17], which is implemented in SymPy (sympy.simplify.fu). In this scheme, trigonometric identities are classified into several groups with names of the form TRi. Some commonly used groups are shown below (rewriting from left to right):

- TR5: $\sin^2 x = 1-\cos^2 x$.
- TR6: $\cos^2 x = 1-\sin^2 x$.
- TR7: $\cos^2 x = \frac{1}{2}(1+\cos 2x)$.
- TR9: $\sin x+\sin y = 2\sin\left(\frac{x+y}{2}\right)\cos\left(\frac{x-y}{2}\right)$, etc.
- TR11: $\sin 2x = 2\sin x\cos x$, $\cos 2x = \cos^2 x-\sin^2 x$, etc.

The **Rewrite trigonometric** rule rewrites using one group of trigonometric identities, followed by simplification. It takes a parameter *rule* which specifies the name of the rule used. For example, applying with *rule* = TR5 on $2-2\sin^2 x$ yields $2\cos^2 x$.

3.3 Substitution

Substitution makes use of the following theorem known from first-year calculus:

$$\int_a^b f(g(x))g'(x)\,dx = \int_{g(a)}^{g(b)} f(u)\,du.$$

There are two possible directions for applying the theorem, corresponding to two rules **Substitution I** and **Substitution II**.

Forward substitution. The rule **Substitution I** assumes the integral is in the form $f(g(x))g'(x)$. Typically in informal writing, only $g(x)$ is provided, and $f(x)$ is found by a sometimes magical process. To see the possible complexity involved, consider the integral

$$\int_{\frac{3}{4}}^{1} \frac{1}{\sqrt{1-x}-1}\,dx$$

The required substitution is $u = \sqrt{1-x}$. The usual explanation continues as follows. Compute $du = -\frac{1}{2}(1-x)^{-1/2}\,dx = -\frac{1}{2}u^{-1}\,dx$. So $dx = -2u \cdot du$. The values of u at the boundary points are $\frac{1}{2}$ and 0. So the integral can be rewritten as $\int_{1/2}^{0} -2u/(u-1)\,du = \int_{0}^{1/2} 2u/(u-1)\,du$.

Heurstic methods are needed for finding a suitable function f. Hence, we require the **Substitution I** rule to specify both f and g as parameters. The rule checks that $f(g(x))g'(x)$ and the original integrand become the same after simplification. We also restrict g to be monotonic (equivalently $g'(x) \geq 0$ or $g'(x) \leq 0$ in the open interval (a, b))[1]. For example, the previous substitution is given by $f(u) = 2u/(u-1)$ and $g(x) = \sqrt{1-x}$.

Backward substitution. The rule **Substitution II** applies substitution in the other direction. In informal writing, it is usually expressed as substituting x by some expression $g(t)$. Then f is the original integrand, but the values of a and b need to be found by the reader. Our rule requires specifying a and b so that $g(a)$ and $g(b)$ equals the original limits of integration, and g is monotonic in the range (a, b). For example, the step

$$\int_{0}^{1} \sqrt{1-x^2}\,dx = \int_{0}^{\frac{\pi}{2}} \sqrt{1-\sin^2 t}\cos t\;dt$$

is represented as $g = \sin(t), a = 0$ and $b = \pi/2$.

3.4 Integration by Parts

The **Integration by parts** rule applies the theorem

$$\int_{a}^{b} u(x)v'(x)\,dx = u(x)v(x)|_{a}^{b} - \int_{a}^{b} u'(x)v(x)\,dx$$

Typically in informal writing, both u and v are provided. These are recorded as parameters of the rule. The rule checks that $f \cdot dx = u \cdot dv$, where f is the original integrand. For example, the step

$$\int_{-1}^{2} xe^x\,dx = xe^x|_{-1}^{2} - \int_{-1}^{2} e^x\,dx$$

is represented as $u = x$ and $v = e^x$.

[1] It is possible to relax this assumption, but the process for reconstructing the proof would be more involved.

3.5 Rewriting

The **Rewrite** rule provides more flexibility for rewriting than simplification. It allows rewriting an expression to any equivalent form as the preparation for applying other rules. The rule takes a parameter *rhs* specifying the intended right side of the rewrite, and another expression *denom*, defaulting to 1. The rule checks that *denom* is nonzero in the domain of integration, and the original expression and *rhs* have the same simplification after multiplying by *denom*.

The presence of *denom* means polynomial division and partial fraction decomposition can be specified. For example, when integrating $x^3/(x^2+1)$, the first step is to divide the numerator by the denominator, yielding $x - x/(x^2+1)$. Simplification as we have implemented is not strongly enough to show their equivalence. However, after multiplying both sides by $\mathit{denom} = x^2+1$, the expressions x^3 and $x(x^2+1) - x$ become the same after simplification.

3.6 Splitting an Integral

Sometimes it is necessary to split the domain of integration into two or more parts. This is needed to deal with absolute values, and non-monotonic functions g in a substitution. The rule **Split region** takes a parameter c satisfying $a \le c \le b$, and split the integral $\int_a^b f(x)\,dx$ into $\int_a^c f(x)\,dx + \int_c^b f(x)\,dx$. For example, when integrating $\int_{-1}^1 \sqrt{x^2}\,dx$ (the example from the introduction), the first step is to split with $c = 0$, resulting in $\int_{-1}^0 \sqrt{x^2}\,dx + \int_0^1 \sqrt{x^2}\,dx$, which can then be simplified to $\int_{-1}^0 -x\,dx + \int_0^1 x\,dx$.

3.7 Solving Equations

One particularly interesting technique for integration involves solving for the value of the integral in an equation[2]. If an integral I can be written in the form $X - cI$, where X is any expression (containing no or simpler integrals), and c is a constant not equal to -1, then we can solve the equation $I = X - cI$ to obtain $I = X/(c+1)$. Common uses of this technique include integrating expressions of the form $e^{ax}\sin bx$ and $e^{ax}\cos bx$ (apply integration by parts twice, then solve equation). The rule **Solve equation** is applied only to the whole expression, and takes two parameters: the index *id* of a previous step and a coefficient *coeff*. Let I be the integral before step *id*. The rule adds $\mathit{coeff} \cdot I$ to the current expression, then divide by $\mathit{coeff}+1$ and simplify. For example, in the evaluation of $\int_0^{\pi/2} e^{2x}\cos x\,dx$, after some steps we get $-2 + e^\pi - 4\int_0^{\pi/2} e^{2x}\cos x\,dx$. Then, applying **Solve equation** with $\mathit{id} = 1$ and $\mathit{coeff} = 4$ yields the answer $\frac{1}{5}(-2+e^\pi)$.

4 User Interface

Above the level of representation of a computation, the graphical user interface helps the user to specify a computation in several ways. Compared to editing a computation directly, the user interface provides the following conveniences:

[2] This is valid as long as the integral exists. In our setting this holds as long as the integrand is continuous.

– Display of all expressions in LaTeX format.
– Selection of actions and subexpressions to perform the action on.
– Automatically generate some parameters of steps.
– Access to automatic integration algorithms such as Slagle's method.

In the remainder of this section, we describe the last two functionalities in more detail. A screenshot of the user interface is shown in Figure 1.

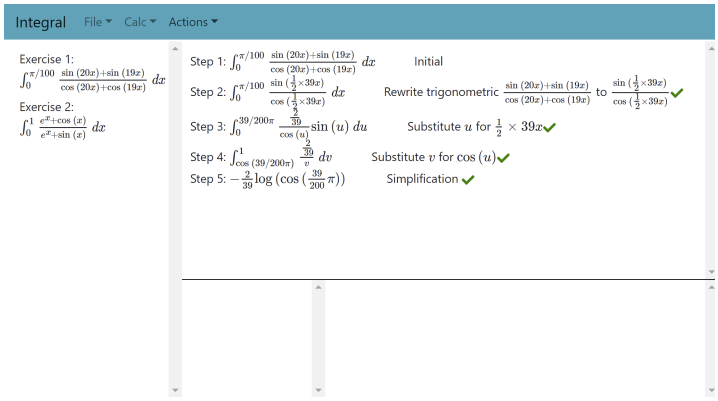


Fig. 1. Screenshot of the user interface, showing the computation of Example 2 in Section 6.

4.1 Substitution

As discussed in Section 3.3, the **Substitution I** rule requires both f and g as parameters, while typically only g is specified in informal arguments. Finding the function f can be a nontrivial process. We try two heuristic methods for finding f. First, if the substitution $u = g(x)$ can be solved for x, yielding a function h such that $x = h(u)$, then f can be found by dividing the integrand by $g'(x)$, then substituting $h(u)$ for x and simplify. Both solving and simplification can be done without checking well-definedness of intermediate expressions, since in the end one only need $f(g(x))g'(x)$ to equal the integrand. For the implementation, we use SymPy's solve function to attempt to find h. The second heuristic simply replaces all expressions equal to $g(x)$ by u, then hope that all remaining occurrences of x is in a single $g'(x)$ in the numerator. Note that the user can always first rewrite the expression into a form where the second heuristic can be applied.

4.2 Rational Functions

Polynomial division or partial fraction decomposition is a common first step for integrating rational functions. From the user interface, the user can in-

voke these actions. Then SymPy's apart method is used to obtain the results, For example, starting from the integral $\int_{1/3}^{1/2} \frac{x}{1-x^4}\,dx$, the user may choose partial fraction decomposition from the menu, which turns the integral into $\int_{1/3}^{1/2} \frac{x}{2(x^2+1)} - \frac{1}{4(x+1)} - \frac{1}{4(x-1)}\,dx$. The **Rewrite** rule with appropriate *denom* parameter is generated from this step.

4.3 Trigonometric Identities

For the application of trigonometric identities, the user does not need to remember names of any rules in Fu's method. Instead, the user selects a subexpression to rewrite. Then, each of Fu's rules are applied in turn using SymPy. In case the application of any rule modifies the expression, the new expression is displayed, and the user can select from the displayed options. The selected action is then recorded with the corresponding name.

4.4 Slagle's Method

We implement a heuristic integration method due to Slagle [45]. There are two main reasons why we choose Slagle's method. First, it is simple but effective for college-level problems. Second, it can output human-readable reasoning steps. This method maintains a search tree consisting of AND-nodes and OR-nodes. Each node contains an integral, with the root containing the original integral. An AND-node specifies that the integral at the node would be solved if each of its child nodes are solved. An OR-node specifies that the integral at the node would be solved if one of its child nodes is solved. The method iteratively expands the tree using a list of *algorithmic* and *heuristic* rules. Algorithmic rules involve basic normalization operations such as simplification and polynomial division, they are always applied to each node. In contrast, heuristic rules are more exploratory, such as guessing potential expressions for substitution, and count as one step in the search.

Our implementation is mostly faithful to the original presentation [45], with some modifications to fit better with our framework. The output of Slagle's method (if successful) is a list of applications of algorithmic and heuristic rules. Each rule can then be converted to one or more computation steps described in Section 3.

5 Proof Translation

We now describe the process for translating a computation to a proof in higher-order logic. This requires sufficiently strong proof automation for verifying the application of each integration rule. The main components of the automation include showing two expressions are equal by simplification, inequality checking, and showing continuity, differentiability, and integrability of functions. The proof automation is implemented in Python based on HolPy. However, it should be possible to implement it in other proof assistants, and one aim of this section is to provide details to facilitate this process.

5.1 Introduction to HolPy

HolPy [49] is a new system for interactive theorem proving implemented in Python. Like Isabelle [32], HOL Light [21], and HOL4 [1], it uses higher-order logic as the logical foundation. The design of HolPy centers around explicit proof terms that can be generated and checked as Python objects, and written to a file in JSON format. Macros are used pervasively to control the size of proof terms. An API for proof automation facilitates implementation of procedures generating proof terms, in a manner similar to writing proof automation in the ML family of languages, but in the setting of an imperative programming language.

5.2 Background Library

For the background library in analysis, we ported statements of over a thousand theorems from HOL Light, of which about 40% are proved using the point-and-click based user interface [49]. However, major parts of the theory are yet to be formalized, including the construction of real numbers, the gauge integral, and the fundamental theorem of calculus. At present, the statements of the theorems need to be trusted. Finishing the formalization of the analysis library is planned as future work.

5.3 Structure of Proof Automation

The procedure for translating a computation is as follows. For each step in the computation, all expressions involved are first translated into terms in higher-order logic. Depending on the rule used, the automation applies the appropriate conversion to the input term, with the parameters of the rule serving as additional arguments to the conversion. Next, the automation attempts to show the equality between the result of the conversion and the expected output of the step by simplifying both sides. Hence, there does not need to be perfect agreement in the expected output and what is computed by proof automation. The translation is successful as long as proof automation is able to show their equivalence. In this way, we allow additional flexibility in the implementations.

We now discuss the overall structure of proof automation, which bears some similarity to the structure of auto and simp tactics in Isabelle [48]. We maintain two tables: a table of proof rules and a table of simplification rules. Each table is indexed by the head of the predicate or term the rule expects. There may be multiple rules associated to the same head term.

- A *prove* rule for a predicate p takes as input a goal whose head is p and a list of assumptions, and attempts to prove the goal. A simple way to specify a prove rule is from a list of theorems whose conclusion matches the given predicate. The corresponding prove rule attempts to apply each of the theorems in order. In case a theorem has assumptions, it recursively applies the overall prove procedure (described below) to discharge each assumption.

– A *simplification* rule for a function f takes as input a term whose head is f and a list of assumptions, and computes the simplification of the term under these assumptions. A simple way to specify a simplification rule is from a list of theorems whose conclusion is an equality, where the left side has head f. The corresponding simplification rule attempts to rewrite using each of the equalities in order. Assumptions in the theorem are discharged by recursive calls to prove as in the previous case.

The overall procedure is defined as a mutual recursion between two functions prove and norm. The norm function receives a term and a list of assumptions as input. It first recursively applies itself to the subterms of the term. Next, it looks for simplification rules associated to the head of the term and applies them in turn. If the head changes, the process is repeated. Note the prove function may be called to discharge assumptions of rewrite rules. This continues until the term is not changed by the simplification rules. The prove function takes a goal and a list of assumptions as input. It first simplifies the goal, then look for prove rules associated to the head term and applies each of them in turn. The case where the goal is an equality reduces to simplifying both sides and then comparing whether they are the same.

5.4 Inequality Checking

A major task of proof automation is checking inequalities in one variable x constrained to lie in an interval $[a, b]$ or (a, b). For example, if one wishes to simplify $\sqrt{f(x)^2}$ to $f(x)$ in the integrand, where the integral is from a to b, one needs to check $f(x) \geq 0$ in the open interval (a, b). Here f may involve the usual arithmetic operations, as well as logarithm, exponential, and trigonometric functions.

The general problem of inequality checking is undecidable when special functions are involved. Hence, we can only hope for methods that can solve most of the inequality goals that appear in practice. There are many heuristic methods [7] as well as decision procedures for inequalities. For our purposes, we found the following, which can be considered as a simplified version of interval arithmetic, to be both simple and effective: starting from the assumption that x lies in a certain interval, iteratively deduce the intervals constraining each of the subterms in the expression. The derivation for each subterm depends on the head of the subterm. Of course, this method is incomplete as it tends to over-approximate the intervals of terms formed from binary operators. Implementation of more advanced inequality checking methods is a goal for the future.

5.5 Simplification

Simplification for arithmetic operations follows the same principle as in Section 3.1: expand the expression into polynomial form, but do not expand powers. We also do not reduce rational functions. This is similar to the normalization of polynomials in other implementations of proof automation [7].

More precisely, define a monomial to be a term of the form $c \cdot (a_1^{p_1} a_2^{p_2} \cdots a_k^{p_k})$, where c is a rational number, and each a_i is either a prime number or a term whose head is not an arithmetic operator. If a_i is a prime number, then the corresponding p_i must be either non-constant or a rational number between 0 and 1 exclusive. The a_i's are distinct and sorted in a pre-determined order. A rational number is a special case of a monomial, with $k = 0$. We call c the coefficient of a monomial and $a_1^{p_1} a_2^{p_2} \cdots a_k^{p_k}$ its body. A polynomial is a sum of monomials, whose bodies are all distinct and in sorted order. It is clear that any expression can be simplified into this form. For example, $\sqrt{6}\sqrt{2}(x + 3^{2/3})$ is simplified to

$$6^{1/2}2^{1/2}x + 6^{1/2}2^{1/2}3^{2/3} = 2^{1/2}3^{1/2}2^{1/2}x + 2^{1/2}3^{1/2}2^{1/2}3^{2/3} = 2 \cdot 3^{1/2}x + 6 \cdot 3^{1/6}$$

Simplification of polynomials is implemented in the simplification rules for $+$, $\times$ and power. $a - b$ and a/b are simply reduced to $a + (-1) \cdot b$ and $a \cdot b^{-1}$, respectively.

For logarithms and exponentials, we apply the standard simplification rules $\log 1 = 0, \log(e^x) = x$ and $e^0 = 1, x > 0 \longrightarrow e^{\log x} = x$. Simplifying trigonometric functions applied to special values is trickier, as we may need to add or subtract multiples of π. For example, $\cos \frac{7\pi}{3}$ is first rewritten to $\cos \frac{\pi}{3}$ and then to $\frac{1}{2}$.

When simplifying an integral over the closed interval $[a, b]$, we apply the following congruence rule:

$$\forall x \in (a, b).\ f(x) = g(x) \longrightarrow \int_a^b f(x)\,dx = \int_a^b g(x)\,dx.$$

This allows us to assume $x \in (a, b)$ when simplifying $f(x)$.

5.6 Applying Theorems

For proving continuity and differentiability, we set up the corresponding prove rules using lists of introduction rules. Some of these rules require assumptions that are discharged recursively. For example, the introduction rule for division is as follows:

$$[\![\ \text{continuous\_on}\ S\ f,\ \text{continuous\_on}\ S\ g,\ \forall x \in S.\ g(x) \neq 0\ ]\!]$$
$$\longrightarrow \text{continuous\_on}\ S\ (\lambda x.\ f(x)/g(x))$$

Application of this rule involves recursively proving the three assumptions, including the use of inequality checking from Section 5.4.

Substitution and integration by parts are implemented by applying the corresponding theorems. This is simple because the parameters of the rule already contain instantiations for all function variables.

6 Evaluation and Examples

We evaluated our prototype implementation[3] on problems taken from exam preparation books (Tongji), online problem lists by D. Kouba [27] (Kouba) and

[3] The code and examples are available online at `https://github.com/bzhan/holpy`.

the MIT Integration Bee [2] (MIT). We also compared our results with Maple and WolframAlpha. Statistics from the evaluation are shown in Table 1.

Problem set	Total	Solved	Ratio	Slagle	Ratio	Maple	WolframAlpha
Tongji	36	36	100%	26	72%	32	35
Kouba/Substitution	18	17	94%	13	72%	18	18
Kouba/Exponentials	12	7	58%	7	58%	12	11
Kouba/Trigonometric	27	22	81%	11	41%	18	22
Kouba/ByParts	23	22	96%	17	74%	23	23
Kouba/LogArcTangent	22	21	95%	13	59%	21	21
Kouba/PartialFraction	20	16	80%	8	40%	18	20
MIT/2013	25	20	80%	14	56%	20	24
Total	183	161	88%	109	60%	162	174

Table 1. Statistics on the problem lists. "Solved" indicates the number of problems for which proofs can be successfully reconstructed from human-provided computations. "Slagle" indicates the number of problems that can be solved by Slagle's method, with successful proof reconstruction. "Maple" represents the number of problems solved by Maple. "WolframAlpha" represents the number of problems which WolframAlpha can give step-by-step solutions without exceeding its time limit.

The Kouba problem lists are divided into different categories based on techniques used. With human-provided computation steps, we can reconstruct proofs for all of the Tongji problems, most of the problems in D. Kouba's list, while problems from the MIT Integration Bee are more challenging (with the later years increasing in difficulty). Most of the failures are due to unable to show equality after simplification, and during inequality checking. Some are due to unsupported functions.

We show two interesting examples from our case studies. SymPy (version 1.5) returns a wrong answer on the first example and times out on the second. The second example takes a long time even for Mathematica, and cannot be solved by its online version WolframAlpha. These examples demonstrate that our system avoids the common errors, and since the user can guide the computation step-by-step, is also able to verify integrals that are difficult even for sophisticated computer algebra systems.

The first example (Tongji, #27) demonstrates the splitting of domain of integration, as well as use of trigonometric identities. The integral is

$$\int_0^\pi \sqrt{1+\cos 2x}\,dx$$

This integral is incorrectly evaluated by SymPy as 0. It is correctly evaluated by Mathematica almost instantly.

The evaluation begins with application of trigonometric identities, rewriting the integrand to $\sqrt{1+\cos^2 x-\sin^2 x}$ and then to $\sqrt{2\cos^2 x}$. For this, the user simply needs to select $\cos 2x$ and then $\sin^2 x$, and choose the desired rewrite targets. The resulting situation is similar to the example given in the introduction. It is then necessary to split the domain of integration where $\cos x = 0$. The system is able to automatically determine $x = \frac{\pi}{2}$. The full computation is:

$$
\begin{aligned}
I &= \int_0^{\pi} \sqrt{1+\cos^2 x - \sin^2 x}\, dx \quad \text{(Rewrite trig. rule TR11)} \\
&= \int_0^{\pi} \sqrt{2\cos^2 x}\, dx = \sqrt{2}\int_0^{\pi} |\cos x|\, dx \quad \text{(Rewrite trig. rule TR5, Simplification)} \\
&= \sqrt{2}\left(\int_0^{\frac{\pi}{2}} |\cos x|\, dx + \int_{\frac{\pi}{2}}^{\pi} |\cos x|\, dx\right) \quad \text{(Split region with } c = \frac{\pi}{2}\text{)} \\
&= 2\sqrt{2} \quad \text{(Elim absolute value, Simplification)}
\end{aligned}
$$

The second example comes from MIT Integration Bee 2019, problem #14:

$$I = \int_0^{\pi/100} \frac{\sin(20x)+\sin(19x)}{\cos(20x)+\cos(19x)}\, dx$$

It is simple if one notices to apply the sum-to-product identity first, but almost impossible otherwise. WolframAlpha fails to find the symbolic answer. Using Mathematica offline, it takes about 15 seconds to return an answer, which is however much more complicated than necessary.

The full computation using our tool is:

$$
\begin{aligned}
I &= \int_0^{\pi/100} \frac{\sin\left(\frac{39}{2}x\right)}{\cos\left(\frac{39}{2}x\right)}\, dx \quad \text{(Rewrite trigonometric, rule TR9)} \\
&= \int_{\cos\left(\frac{39\pi}{200}\right)}^{1} \frac{2}{39}\frac{1}{t}\, dt \quad \text{(Substitution I with } g = \cos\left(\frac{39}{2}x\right)\text{)} \\
&= -\frac{2}{39}\log\left(\cos\frac{39\pi}{200}\right) \quad \text{(Simplification).}
\end{aligned}
$$

7 Conclusion

In this paper, we proposed a framework for verifying symbolic computation of definite integrals, where the user can perform computations in an interface familiar from computer algebra systems, but with results verified by automatic translation to proofs in higher-order logic. The design of the framework follows a layered approach, with each layer focusing on a different aspect of the problem: methods for solving integrals, computer algebra, and proof reconstruction. We implemented a prototype system based on HolPy, and evaluated it on a test suite consisting of publicly available problem lists at the undergraduate level, showing its effectiveness on a large majority of cases.

One immediate piece of future work is to secure the foundation of the higher-order logic proof, by formalizing the proofs of the required theorems. Another gap is the arithmetic computation and comparison of real constants, which, in the case of comparisons, would require approximation techniques [10].

Our prototype implementation focuses on definite integrals of one-variable functions. However, the idea can be applied more generally, by suitably extending the language of integration rules. For applications in the engineering domain, some extensions that would be of high value include linear algebra, improper integrals (including Laplace and Fourier transforms), and vector calculus.

References

1. The HOL 4 system. http://hol.sourceforge.net/
2. MIT Integration Bee. `http://www.mit.edu/~pax/integrationbee.html`, accessed: 2020-1-22
3. Adams, A., Dunstan, M., Gottliebsen, H., Kelsey, T., Martin, U., Owre, S.: Computer algebra meets automated theorem proving: Integrating Maple and PVS. In: Boulton, R.J., Jackson, P.B. (eds.) Theorem Proving in Higher Order Logics. Lecture Notes in Computer Science, vol. 2152, pp. 27–42. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)
4. Althoff, M., Frehse, G., Girard, A.: Set propagation techniques for reachability analysis. Annual Review of Control, Robotics, and Autonomous Systems **4**(1) (2021)
5. Arias, E.J.G., Pin, B., Jouvelot, P.: jsCoq: Towards hybrid theorem proving interfaces. In: Autexier, S., Quaresma, P. (eds.) Proceedings of the 12th Workshop on User Interfaces for Theorem Provers, UITP 2016, Coimbra, Portugal, 2nd July 2016. EPTCS, vol. 239, pp. 15–27 (2016)
6. Aström, K.J., Murray, R.M.: Feedback Systems: An Introduction for Scientists and Engineers. Princeton University Press, Princeton (2008)
7. Avigad, J., Lewis, R.Y., Roux, C.: A heuristic prover for real inequalities. J. Autom. Reasoning **56**(3), 367–386 (2016)
8. Ballarin, C., Homann, K., Calmet, J.: Theorems and algorithms: An interface between Isabelle and Maple. In: Levelt, A.H.M. (ed.) Proceedings of the 1995 International Symposium on Symbolic and Algebraic Computation. p. 150157. ISSAC '95, Association for Computing Machinery, New York, NY, USA (1995)
9. Bohrer, B., Rahli, V., Vukotic, I., Völp, M., Platzer, A.: Formally verified differential dynamic logic. In: Bertot, Y., Vafeiadis, V. (eds.) Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, Paris, France, January 16-17, 2017. pp. 208–221 (2017)
10. Bréhard, F., Mahboubi, A., Pous, D.: A certificate-based approach to formally verified approximations. In: Harrison, J., O'Leary, J., Tolmach, A. (eds.) 10th International Conference on Interactive Theorem Proving, ITP 2019, September 9-12, 2019, Portland, OR, USA. LIPIcs, vol. 141, pp. 8:1–8:19 (2019)
11. Buchberger, B., Jebelean, T., Kutsia, T., Maletzky, A., Windsteiger, W.: Theorema 2.0: Computer-assisted natural-style mathematics. J. Formaliz. Reason. **9**(1), 149–185 (2016)
12. Butler, R.W.: Formalization of the integral calculus in the PVS theorem prover. J. Formalized Reasoning **2**(1), 1–26 (2009)
13. Chen, S., Wang, G., Li, X., Zhang, Q., Shi, Z., Guan, Y.: Formalization of camera pose estimation algorithm based on rodrigues formula. Formal Aspects Comput. **32**(4-6), 417–437 (2020)
14. Chyzak, F., Mahboubi, A., Sibut-Pinote, T., Tassi, E.: A computer-algebra-based formal proof of the irrationality of $\zeta(3)$. In: Klein, G., Gamboa, R. (eds.) Interactive Theorem Proving. Lecture Notes in Computer Science, vol. 8558, pp. 160–176. Springer International Publishing, Cham (2014)
15. Durán, A.J., Pérez, M., Varona, J.L.: The misfortunes of a trio of mathematicians using computer algebra systems. can we trust in them? Notices Amer. Math. Soc. **61**(10), 1249–1252 (2014)
16. Eberl, M.: Verified real asymptotics in Isabelle/HOL. In: Davenport, J.H., Wang, D., Kauers, M., Bradford, R.J. (eds.) Proceedings of the 2019 on International

Symposium on Symbolic and Algebraic Computation, ISSAC 2019, Beijing, China, July 15-18, 2019. pp. 147–154. ACM (2019)
17. Fu, H., Zhong, X., Zeng, Z.: Automated and readable simplification of trigonometric expressions. Mathematical and Computer Modelling **44**(11-12), 1169–1177 (2006)
18. Fulton, N., Mitsch, S., Quesel, J., Völp, M., Platzer, A.: KeYmaera X: an axiomatic tactical theorem prover for hybrid systems. In: Felty, A.P., Middeldorp, A. (eds.) Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9195, pp. 527–538 (2015)
19. Geddes, K.O., Czapor, S.R., Labahn, G.: The Risch Integration Algorithm, pp. 511–573. Springer US, Boston, MA (1992)
20. Harrison, J.: Theorem proving with the real numbers. CPHC/BCS distinguished dissertations, Springer (1998)
21. Harrison, J.: HOL Light: An overview. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) Theorem Proving in Higher Order Logics. Lecture Notes in Computer Science, vol. 5674, pp. 60–66. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
22. Harrison, J.: Formal proofs of hypergeometric sums - dedicated to the memory of Andrzej Trybulec. J. Autom. Reasoning **55**(3), 223–243 (2015)
23. Harrison, J., Théry, L.: A skeptic's approach to combining HOL and Maple. J. Autom. Reason. **21**(3), 279–294 (1998)
24. Hölzl, J., Heller, A.: Three chapters of measure theory in Isabelle/HOL. In: van Eekelen, M., Geuvers, H., Schmaltz, J., Wiedijk, F. (eds.) Interactive Theorem Proving - Second International Conference, ITP 2011, Berg en Dal, The Netherlands, August 22-25, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6898, pp. 135–151 (2011)
25. Immler, F.: A verified ODE solver and the Lorenz attractor. J. Autom. Reason. **61**(1-4), 73–111 (2018)
26. Immler, F., Traut, C.: The flow of ODEs. In: Blanchette, J.C., Merz, S. (eds.) Interactive Theorem Proving - 7th International Conference, ITP 2016, Nancy, France, August 22-25, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9807, pp. 184–199 (2016)
27. Kouba, D.A.: The calculus page problems list. `https://www.math.ucdavis.edu/~kouba/ProblemsList.html`, accessed: 2020-1-22
28. Lewis, R.Y.: An extensible ad hoc interface between Lean and Mathematica. In: Dubois, C., Paleo, B.W. (eds.) Proceedings of the Fifth Workshop on Proof eXchange for Theorem Proving, PxTP 2017, Brasília, Brazil, 23-24 September 2017. EPTCS, vol. 262, pp. 23–37 (2017)
29. Li, L., Shi, Z., Guan, Y., Zhang, Q., Li, Y.: Formalization of geometric algebra in HOL Light. J. Autom. Reasoning **63**(3), 787–808 (2019)
30. Mahboubi, A., Melquiond, G., Sibut-Pinote, T.: Formally verified approximations of definite integrals. J. Autom. Reason. **62**(2), 281–300 (2019)
31. Mhamdi, T., Hasan, O., Tahar, S.: On the formalization of the Lebesgue integration theory in HOL. In: Kaufmann, M., Paulson, L.C. (eds.) Interactive Theorem Proving, First International Conference, ITP 2010, Edinburgh, UK, July 11-14, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6172, pp. 387–402 (2010)
32. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL - A Proof Assistant for Higher-Order Logic, Lecture Notes in Computer Science, vol. 2283. Springer (2002)

33. Oppenheim, A.V., Willsky, A.S.: Signals and Systems. Prentice Hall, Upper Saddle River, New Jersey (1996)
34. Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom. Reason. **41**(2), 143–189 (2008)
35. Platzer, A.: A complete uniform substitution calculus for differential dynamic logic. J. Autom. Reason. **59**(2), 219–265 (2017)
36. Prajna, S., Jadbabaie, A.: Safety verification of hybrid systems using barrier certificates. In: Alur, R., Pappas, G.J. (eds.) Hybrid Systems: Computation and Control, 7th International Workshop, HSCC 2004, Philadelphia, PA, USA, March 25-27, 2004, Proceedings. Lecture Notes in Computer Science, vol. 2993, pp. 477–492 (2004)
37. Rashid, A., Hasan, O.: On the formalization of Fourier transform in higher-order logic. In: Blanchette, J.C., Merz, S. (eds.) Interactive Theorem Proving. Lecture Notes in Computer Science, vol. 9807, pp. 483–490. Springer International Publishing, Cham (2016)
38. Rashid, A., Hasan, O.: Formal analysis of continuous-time systems using Fourier transform. J. Symb. Comput. **90**, 65–88 (2019)
39. Rich, A.D., Scheibe, P., Abbasi, N.M.: Rule-based integration: An extensive system of symbolic integration rules. J. Open Source Softw. **3**(32), 1073 (2018)
40. Richter, S.: Formalizing integration theory with an application to probabilistic algorithms. In: Slind, K., Bunker, A., Gopalakrishnan, G. (eds.) Theorem Proving in Higher Order Logics, 17th International Conference, TPHOLs 2004, Park City, Utah, USA, September 14-17, 2004, Proceedings. Lecture Notes in Computer Science, vol. 3223, pp. 271–286 (2004)
41. Sankaranarayanan, S., Sipma, H., Manna, Z.: Constructing invariants for hybrid systems. In: Alur, R., Pappas, G.J. (eds.) Hybrid Systems: Computation and Control, 7th International Workshop, HSCC 2004, Philadelphia, PA, USA, March 25-27, 2004, Proceedings. Lecture Notes in Computer Science, vol. 2993, pp. 539–554 (2004)
42. Selsam, D., Liang, P., Dill, D.L.: Developing bug-free machine learning systems with formal mathematics. In: Precup, D., Teh, Y.W. (eds.) Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017. Proceedings of Machine Learning Research, vol. 70, pp. 3047–3056 (2017)
43. Shi, Z., Gu, W., Li, X., Guan, Y., Ye, S., Zhang, J., Wei, H.: The gauge integral theory in HOL4. J. Applied Mathematics **2013**, 160875:1–160875:7 (2013)
44. Shi, Z., Wu, A., Yang, X., Guan, Y., Li, Y., Song, X.: Formal analysis of the kinematic Jacobian in screw theory. Formal Aspects Comput. **30**(6), 739–757 (2018)
45. Slagle, J.R.: A heuristic program that solves symbolic integration problems in freshman calculus. J. ACM **10**(4), 507–520 (1963)
46. Taqdees, S.H., Hasan, O.: Formalization of Laplace transform using the multivariable calculus theory of HOL-Light. In: McMillan, K., Middeldorp, A., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14-19, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8312, pp. 744–758 (2013)
47. Wang, S., Zhan, N., Zou, L.: An improved HHL prover: An interactive theorem prover for hybrid systems. In: Butler, M., Conchon, S., Zaïdi, F. (eds.) Formal Methods and Software Engineering - 17th International Conference on Formal Engineering Methods, ICFEM 2015, Paris, France, November 3-5, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9407, pp. 382–399 (2015)

48. Wenzel, M.: The Isabelle/Isar reference manual. http://isabelle.in.tum.de/doc/isar-ref.pdf
49. Zhan, B., Ji, Z., Zhou, W., Xiang, C., Hou, J., Sun, W.: Design of point-and-click user interfaces for proof assistants. In: Ait-Ameur, Y., Qin, S. (eds.) Formal Methods and Software Engineering - 21st International Conference on Formal Engineering Methods, ICFEM 2019, Shenzhen, China, November 5-9, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11852, pp. 86–103 (2019)