Noname manuscript No.
(will be inserted by the editor)

Formally Verified Approximations of Definite Integrals

Assia Mahboubi · Guillaume Melquiond · Thomas Sibut-Pinote

the date of receipt and acceptance should be inserted later

Abstract Finding an elementary form for an antiderivative is often a difficult task, so numerical integration has become a common tool when it comes to making sense of a definite integral. Some of the numerical integration methods can even be made rigorous: not only do they compute an approximation of the integral value but they also bound its inaccuracy. Yet numerical integration is still missing from the toolbox when performing formal proofs in analysis.

This paper presents an efficient method for automatically computing and proving bounds on some definite integrals inside the Coq formal system. Our approach is not based on traditional quadrature methods such as Newton-Cotes formulas. Instead, it relies on computing and evaluating antiderivatives of rigorous polynomial approximations, combined with an adaptive domain splitting. Our approach also handles improper integrals, provided that a factor of the integrand belongs to a catalog of identified integrable functions. This work has been integrated to the CoqInterval library.

Keywords Formal proof · Numeric computations · Definite integrals · Improper integrals · Decision procedure · Interval arithmetic · Polynomial approximations · Real analysis

This work was supported in part by the project FastRelax ANR-14-CE25-0018-01.

A. Mahboubi
Inria, LS2N, Université de Nantes
LS2N, 2 rue de la Houssinière, BP 92208 44322 Nantes Cedex 3, France
E-mail: assia.mahboubi@inria.fr

G. Melquiond
Inria, Université Paris-Saclay
PCRI, Bât. 650, Université Paris-Sud, 91405 Orsay cedex, France
E-mail: guillaume.melquiond@inria.fr

T. Sibut-Pinote
École Polytechnique, Inria, Université Paris-Saclay
1 Rue Honoré d'Estienne d'Orves, 91120 Palaiseau, France
E-mail: thomas.sibut-pinote@inria.fr

1 Introduction

Computing the value of definite integrals is the modern and generalized take on the ancient problem of computing the area of a figure. *Quadrature methods* hence refer to the numerical methods for estimating such integrals. Numerical integration is often the preferred way of obtaining such estimations as symbolic approaches may be too difficult or even just impossible. These quadrature methods usually consist in interpolating the integrand function by a degree-n polynomial, integrating the polynomial and then bounding the error using a bound on the n+1-th derivative of the integrand function. Most often though, these methods are used in a non-rigorous way, for instance without bounding the error, or worse on functions with unbounded derivatives. Open formulas of quadrature can also be used to approximate improper integrals with removable singularities, like $\int_0^1 \frac{\sin t}{t} dt$, but their use in practice is even less rigorous.

Yet estimating the value of integrals is a crucial part of some mathematical proofs, making numerical integration an invaluable ally. Examples of such proofs occur in various areas of mathematics, such as number theory (*e.g.* Helfgott's proof of the ternary Goldbach conjecture [7]) or geometry (*e.g.* the first proof of the double bubble conjecture [6]). This motivates developing high-confidence methods for computing *reliable* yet accurate and fast estimations of integrals.

The present article describes a formal-proof producing procedure to obtain numerical enclosures of definite integrals $\int_u^v f$, where f is a real-valued function. It extends a previous publication by the same authors [9], devoted to the case of a bounded integration domain, for an integrand function f which is *Riemann-integrable* on $[u; v]$. This extended version includes a generalization of the enclosure method to the case of *improper integrals*. Improper integrals are limits of definite integrals: for instance, $\int_u^{+\infty} f$ is the limit of $\int_u^v f$ when $v \to +\infty$, and $\int_{a^+}^v f$, with a a singular point for f, denotes the limit of $\int_u^v f$ when $u \to a^+$. Estimating an improper integral amounts to combining two enclosures: one for a proper integral and one for a remainder.

Our procedure can deal with any proper integral of a function f for which we have an interval extension and/or a polynomial approximation. Regarding improper integrals, the current procedure can only deal with a limited class of integrals: their limit bounds should be either 0^+ or $+\infty$, and the syntactic shape of the integrand f should make manifest its domination by a suitable element of the scale $x^\alpha \ln^\beta x$ or of the scale $e^{\gamma x}$. Enclosures are computed *inside* the Coq proof assistant and the computations are correct by construction. Interestingly, the formal proof that the integral exists comes as a by-product of these computations, even in the case of improper integrals.

Our approach is based on interval methods, in the spirit of Moore et al. [13], and combines the computation of a numerical enclosure of the integrand with an adaptive dichotomy process. It is based on the CoqInterval library for computing interval extensions of elementary mathematical functions and is implemented as an improvement of the `interval` Coq tactic [11]. We use the theory of the Riemann integral from the Coquelicot library [3]. The latter is a conservative extension of the theory shipped with the standard distribution of the Coq system: based on the same axiomatic definition of real numbers, the Coquelicot library provides a more comprehensive and user-friendly formal library of real analysis. Note that, for the purpose of the present work, we had to significantly extend the Coquelicot library to improve its support for improper integrals.

The paper is organized as follows: Section 2 introduces some definitions and notations used throughout the paper, and briefly describes the Coq libraries we build on. Section 3 describes the algorithms used to estimate proper integrals while Section 4 focuses on estimating the remainder of improper integrals. Section 5 describes the design of the proof-producing Coq tactic. In Section 6 we provide cross-software benchmarks highlighting issues with both our and others' algorithms. In Section 7, we discuss the limitations and perspectives of this work.

2 Preliminaries

In this section we introduce some vocabulary and notations used throughout the paper and we summarize the existing Coq libraries the present work builds on.

2.1 Notations and first definitions

In this paper, an *interval* is a closed connected subset of the set of real numbers. We use $\mathbb{I}$ to denote the set of intervals: $\{[a; b] \mid a, b \in \mathbb{R} \cup \{\pm\infty\}\}$. A *point interval* is an interval of the shape $[a; a]$ where $a \in \mathbb{R}$. Any interval variable will be denoted using a bold font. For any interval $\mathbf{x} \in \mathbb{I}$, $\inf \mathbf{x}$ (resp. $\sup \mathbf{x}$) denotes its left (resp. right) bound, with $\inf \mathbf{x} \in \mathbb{R} \cup \{-\infty\}$ (resp. $\sup \mathbf{x} \in \mathbb{R} \cup \{+\infty\}$). An *enclosure* of $x \in \mathbb{R}$ is an interval $\mathbf{x} \in \mathbb{I}$ such that $x \in \mathbf{x}$.

Interval arithmetic is concerned with providing operators on intervals that respect the *inclusion property*. Given a binary operator $\diamond$ on real numbers, naive interval arithmetic provides a binary operator $\Diamond$ on intervals such that

$$\forall x, y \in \mathbb{R},\ \forall \mathbf{x}, \mathbf{y} \in \mathbb{I},\ x \in \mathbf{x} \wedge y \in \mathbf{y} \Rightarrow x \diamond y \in \mathbf{x} \Diamond \mathbf{y}.$$

In the following, we will not denote interval operators in any distinguishing way. In particular, whenever an arithmetic operator takes interval inputs, it should be understood as any interval extension of the corresponding operator on real numbers. Moreover, whenever a real number appears as an input of an interval operator, it should be understood as any interval that encloses this number. For instance, the expression $(v - u) \cdot \mathbf{x}$ denotes the interval product of the interval $\mathbf{x}$ with any (hopefully tight) interval enclosing the real $v - u$.

2.2 Elementary real analysis in Coq

Coq's standard library `Reals`[1] axiomatizes real arithmetic, with a classical flavor [12]. It provides some notions of elementary real analysis, including the definition of continuity, differentiability, and Riemann integrability. It also comes with a formalization of the properties of usual mathematical functions like sin, cos, exp, and so on.

[1] https://coq.inria.fr/distrib/current/stdlib/

The Coquelicot library is a conservative extension of this library [3]. Given $\mathbb{V}$ a complete normed $\mathbb{R}$-vector space, *i.e.* an instance of `(CompleteNormedModule R)`, it provides a *total* operator that outputs a value in $\mathbb{V}$ from a function $f : \mathbb{R} \to \mathbb{V}$ and two bounds $u, v \in \mathbb{R}$:

```
Definition RInt {V : CompleteNormedModule R} (f : R -> V) (u : R) (v : R) : V.
```

When the function f is Riemann-integrable on $[u; v]$, the value `(RInt f u v)` is equal to $\int_u^v f(t)\,dt$. Otherwise it is left unspecified. Thus, most properties about the actual value of `(RInt f u v)` hold only if f is integrable on $[u; v]$.

The library also provides a total operator that generalizes the notion of integral by replacing the bounds by filters, which are collections of neighborhoods of the intended finite or infinite bound. The resulting generalized definition of integral can be used to represent improper integrals such as $\int_{0^+}^{+\infty} \ln x/(1+x^2)dx$.

```
Definition RInt_gen {V : ... } (f : R -> V) (u v : (R -> Prop) -> Prop) : V.
```

The aim of this work is to provide a procedure that computes a numerical and formally proved enclosure of an expression `(RInt f u v)` or `(RInt_gen f u v)` —and justifies that this integral is well-defined. This procedure is used in an automated tactic that proves inequalities like $|\int_0^1 \sqrt{1-x^2}\,dx - \frac{\pi}{4}| \le \frac{1}{100}$, stated as:

```
Goal Rabs (RInt (fun x => sqrt(1 - x * x)) 0 1 - PI / 4) <= 1/100.
```

2.3 Numerical computations in Coq

CoqInterval is a Coq library for computing numerical enclosures of real-valued expressions [11]. These expressions belong to a class $\mathcal{E}$ built from constants, variables, arithmetic operations, and some elementary functions. It also provides a tactic `interval` to automatically deduce certain goals from these enclosures.

The tactic typically takes a goal $A \le e \le B$ where e is an expression in $\mathcal{E}$, and A and B are constants. Using the paradigm of interval arithmetic, it builds a set $\mathbf{e}$ such that $e \in \mathbf{e}$ holds by construction and such that $\mathbf{e}$ reduces to an interval $[\inf \mathbf{e}; \sup \mathbf{e}]$ by computation. Then it checks that $A \le \inf \mathbf{e}$ and $\sup \mathbf{e} \le B$, again by computation, from which it proves $A \le e \le B$. All the computations on interval bounds are performed using a rigorous yet efficient formalization of multi-precision floating-point arithmetic.

The inclusion property of interval arithmetic is easily transported from operators to whole expressions by induction on these expressions. This gives a way to obtain the property $e \in \mathbf{e}$ above when $\mathbf{e}$ is built using interval operators. This approach, however, cannot keep track of correlations between subexpressions and might compute overestimated enclosures which are thus useless for proving some goals. For instance, assume that $x \in [3; 4]$, so $-x \in [-4; -3]$ using the interval extension of the negation, so $x + (-x) \in [3 + (-4); 4 + (-3)]$ using the interval extension of the addition. If one wants to prove that $x - x$ is always 0, the interval $[-1; 1]$ obtained by naive interval arithmetic is useless. This is why the CoqInterval library also comes with refinements of naive interval arithmetic, such as automatic differentiation and rigorous polynomial approximations using Taylor models, so as to reduce this loss of correlations.

The goal of this work is to extend the class $\mathcal{E}$ of supported expressions with integrals whose bodies are in $\mathcal{E}$.

3 Interval methods to approximate a proper integral

In this section, we describe how to compute a numerical enclosure of the real number $\int_u^v f$ from enclosures of the finite bounds u and v and of the integrand function f. We describe two basic methods based respectively on the evaluation of a simple interval extension and on a polynomial approximation of f. They can be combined and improved by a dichotomy process.

3.1 Naive integral enclosure

Our first approach uses an *interval extension* of the integrand.

Definition 1 For any function $f : \mathbb{R}^n \to \mathbb{R}$, a function $F : \mathbb{I}^n \to \mathbb{I}$ is an *interval extension* of f on $\mathbb{R}$ if

$$\forall \mathbf{x}_1, \ldots, \mathbf{x}_n,\ \{f(x_1, \ldots, x_n) \mid \forall i, x_i \in \mathbf{x}_i\} \subseteq F(\mathbf{x}_1, \ldots, \mathbf{x}_n).$$

In the rest of the section we suppose that $F : \mathbb{I} \to \mathbb{I}$ is an interval extension of the univariate function f, and we want to compute an enclosure of $\int_u^v f$, with $u, v \in \mathbb{R}$, and f integrable on $[u; v]$.

Definition 2 The *closed convex hull* of a set $A \subseteq \mathbb{R}$ is the smallest interval containing A, denoted here $\mathrm{hull}(A)$. Moreover, the interval $\mathrm{hull}(\mathbf{a}, \mathbf{b})$ denotes the convex hull of (the union of) two intervals $\mathbf{a}$ and $\mathbf{b}$. Finally, $\mathrm{hull}(\mathbf{a}, +\infty)$ designates the interval $[\inf \mathbf{a}; +\infty)$.

Lemma 1 (Naive integral enclosure)

$$\int_u^v f \in (v - u) \cdot \mathrm{hull}\{f(t) \mid t \in [u; v] \vee t \in [v; u]\}. \tag{1}$$

Proof Let us first suppose that $u \le v$. Denote $f([u; v]) := \{f(t) \mid t \in [u; v]\}$. Assume without loss of generality that $f([u; v])$ is bounded. If $[m; M] := \mathrm{hull}(f([u; v]))$, then for any $t \in [u; v]$, we have $m \le f(t) \le M$. So $(v - u)m \le \int_u^v f \le (v - u)M$, hence (1). The case $v \le u$ is symmetrical.

In practice we do not compute with f but only its interval extension F. Moreover, we want the computations to operate using only enclosures of the bounds. So we adapt Formula (1) accordingly.

Lemma 2 (Interval naive integral enclosure) *For any intervals $\mathbf{u}, \mathbf{v}$ such that $u \in \mathbf{u}$ and $v \in \mathbf{v}$, we have*

$$\int_u^v f \in (\mathbf{v} - \mathbf{u}) \cdot F(\mathrm{hull}(\mathbf{u}, \mathbf{v})). \tag{2}$$

Note that if $\mathbf{u}$ and $\mathbf{v}$ are point intervals and if F is the optimal interval extension of f, then (2) reduces to (1).

Proof If $u \in \mathbf{u}$ and $v \in \mathbf{v}$, then by (1) and reusing notations from the proof, we have $\int_u^v f \in (v-u) \cdot \mathrm{hull}(f([u;v]))$. Since $(v-u) \in (\mathbf{v}-\mathbf{u})$, we only have to show that $\mathrm{hull}(f([u;v])) \subseteq F(\mathrm{hull}(\mathbf{u},\mathbf{v}))$. Since $[u;v] \subseteq \mathrm{hull}(\mathbf{u},\mathbf{v})$ and F is an interval extension of f, we have $f([u;v]) \subseteq f(\mathrm{hull}(\mathbf{u},\mathbf{v})) \subseteq F(\mathrm{hull}(\mathbf{u},\mathbf{v}))$. Therefore $\mathrm{hull}(f([u;v]))$ is included in the interval $F(\mathrm{hull}(\mathbf{u},\mathbf{v}))$, by definition of the closed convex hull.

The `naive_integral` Coq function implements (2). Given $\mathbf{u},\mathbf{v} \in \mathbb{I}$ and F a function of type $\mathbb{I} \to \mathbb{I}$, `(naive_integral prec F u v)` computes an interval $\mathbf{i}$ using floating-point arithmetic at precision `prec`. If F is an interval extension of f, if $u \in \mathbf{u}$ and $v \in \mathbf{v}$, and if f is integrable on $[u;v]$, then $\int_u^v f \in \mathbf{i}$.

```
Definition naive_integral prec F u v :=
  𝕀.mul prec (F (𝕀.join u v)) (𝕀.sub prec v u).
```

3.2 Polynomial approximation

The enclosure method described in Section 3.1 is crude. Better knowledge of the integrated function allows for a more efficient approach.

The CoqInterval library defines a *rigorous polynomial approximation* (RPA) of $f : \mathbb{R} \to \mathbb{R}$ on the interval $\mathbf{x}$ as a pair $(\mathbf{p}, \boldsymbol{\Delta})$, with $\mathbf{p} \in \mathbb{I}[X]$, such that there exists a polynomial $p \in \mathbb{R}[X]$ enclosed[2] in $\mathbf{p}$ for which $f(x) - p(x) \in \boldsymbol{\Delta}$ for all $x \in \mathbf{x}$. CoqInterval computes these RPAs by composing and performing arithmetic operations on Taylor expansions of elementary functions [11]. Thanks to these polynomial approximations, we can make use of the following lemma.

Lemma 3 (Polynomial approximation) *Suppose f is approximated on $[u;v]$ by $p \in \mathbb{R}[X]$ and $\boldsymbol{\Delta} \in \mathbb{I}$ in the sense that $\forall x \in [u;v],\ f(x) - p(x) \in \boldsymbol{\Delta}$. Then for any primitive P of p, we have $\int_u^v f \in P(v) - P(u) + (v-u) \cdot \boldsymbol{\Delta}$.*

Proof We have $\int_u^v f - (P(v) - P(u)) = \int_u^v (f(t) - p(t))\, dt$. By hypothesis, the constant function $\boldsymbol{\Delta}$ is an interval extension of $t \mapsto f(t) - p(t)$ on $[u;v]$, hence Lemma 1 applies (notice that $\mathrm{hull}(\boldsymbol{\Delta}) = \boldsymbol{\Delta}$).

Note that our method and proofs do not depend on the way RPAs are obtained. In particular, we are not taking advantage of the fact that p is computed with respect to the center of $[u;v]$, which would make it possible to skip half of the computations [4].

3.3 Quality of the integral enclosures

Both methods described in Sections 3.1 and 3.2 use a single approximation of the integrand on the integration interval. A decomposition of this interval into smaller pieces may increase the accuracy of the enclosure, if tighter approximations are obtained on each subinterval. In this section we give an intuition of how the

[2] We say that $\mathbf{p} \in \mathbb{I}[X]$ is an enclosure of $p \in \mathbb{R}[X]$ if, for all $i \in \mathbb{N}$, the i-th coefficient $\mathbf{p}_i$ of $\mathbf{p}$ is an enclosure of the i-th coefficient p_i of p, where we take the convention that for $i > \deg \mathbf{p}$, $\mathbf{p}_i = \{0\}$ and for $i > \deg p$, $p_i = 0$.

naive and polynomial approaches compare, from a time complexity point of view. The naive (resp. polynomial) approach here consists in using a simple interval approximation (resp. a valid polynomial approximation) to estimate the integral on each subinterval. Let us suppose that we split the initial integration interval, using the interval additivity property of integrals, before computing integral enclosures:

$$\int_u^v f = \int_{x_0}^{x_1} f + \ldots + \int_{x_{n-1}}^{x_n} f \quad \text{with } x_i = u + \tfrac{i}{n}(v-u).$$

Let $w(\mathbf{x}) = \sup \mathbf{x} - \inf \mathbf{x}$ denote the width of an interval. The smaller $w(\mathbf{x})$ is, the more accurately any real $x \in \mathbf{x}$ is approximated by $\mathbf{x}$. Any sensible interval arithmetic respects $w(\mathbf{x}+\mathbf{y}) \simeq w(\mathbf{x}) + w(\mathbf{y})$ and $w(k \cdot \mathbf{x}) \simeq k \cdot w(\mathbf{x})$.

We consider the case of the naive approach first. We assume that F is an optimal interval extension of f and that f has a Lipschitz constant equal to k_0, that is, $w(F(\mathbf{x})) \simeq k_0 \cdot w(\mathbf{x})$. Since $w(\text{naive}([x_i; x_{i+1}])) \simeq (x_{i+1} - x_i) \cdot w(F([x_i; x_{i+1}]))$, we get the following accuracy when computing the integral:

$$w\left(\sum_i \text{naive}([x_i; x_{i+1}])\right) \simeq k_0 \cdot (v-u)^2/n.$$

To gain one bit of accuracy, we need to go from n to $2n$ integrals, which means multiplying the computation time by two, hence an exponential complexity.

Now for the polynomial enclosure. Let us assume we can compute a polynomial approximation of f on any interval $\mathbf{x}$ with an error $\boldsymbol{\Delta}(\mathbf{x})$. We can expect this error to satisfy $w(\boldsymbol{\Delta}(\mathbf{x})) \simeq k_d \cdot w(\mathbf{x})^{d+1}$ with d the degree of the polynomial approximation and k_d depending on f. Since $w(\text{poly}([x_i; x_{i+1}])) \simeq (x_{i+1} - x_i) \cdot w(\boldsymbol{\Delta}([x_i, x_{i+1}]))$, the accuracy is now

$$w\left(\sum_i \text{poly}([x_i; x_{i+1}])\right) \simeq k_d \cdot (v-u)^{d+2}/n^{d+1}.$$

For a fixed d, one still has to increase n exponentially with respect to the target accuracy. The power coefficient, however, is much smaller than for the naive method. By doubling the computation time, one gets $d+1$ additional bits of accuracy.

In order to improve the accuracy of the result, one can increase d instead of n. If f behaves similarly to exp or sin, Taylor-Lagrange formula tells us that k_d decreases as fast as $(d!)^{-1}$. Moreover, the time complexity of computing a polynomial approximation usually grows like d^3. So, if $n \simeq v - u$, doubling the computation time by increasing d gives about 25% more bits of accuracy.

As can be seen from the considerations above, striking the proper balance between n and d for reaching a target accuracy in a minimal amount of time is difficult, so we have made the decision of letting the user control d (see Section 5.3) while the implementation adaptively splits the integration interval. Had we not been constrained by Coq's logic, we could have accessed a clock so as to dynamically balance between n and d [4].

3.4 Dichotomy and adaptivity

Both methods presented in Sections 3.1 and 3.2 can compute an interval enclosing $\int_u^v f$ when u and v are proper bounds. Polynomial approximations usually give tighter enclosures of the integral, but not always, so we combine both methods by taking the intersection of their result.

This may still not be sufficient for getting a tight enough enclosure, in which case we recursively split the integration domain in two parts, using the interval additivity property of integral. The function `integral_interval_absolute` performs this dichotomy and the integration on each subdomain. It takes an absolute error parameter ε; it stops splitting as soon as the width of the computed integral enclosure is smaller than ε. The function also takes a *depth* parameter, which means that the initial domain is split into at most $2^{depth+1}$ subdomains. Note that, because the depth is bounded, there is no guarantee that the target width will be reached.

Let us detail more precisely how the function behaves. It starts by splitting $[u; v]$ into $[u; m]$ and $[m; v]$ where $m = \frac{u+v}{2}$. It then computes some enclosures $\mathbf{i_1}$ of $\int_u^m f$ and $\mathbf{i_2}$ of $\int_m^v f$. If $depth = 0$, the function returns $\mathbf{i_1} + \mathbf{i_2}$. Otherwise, several cases can occur:

- If $w(\mathbf{i_1}) \leq \frac{\varepsilon}{2}$ and $w(\mathbf{i_2}) \leq \frac{\varepsilon}{2}$, the function simply returns $\mathbf{i_1} + \mathbf{i_2}$.
- If $w(\mathbf{i_1}) \leq \frac{\varepsilon}{2}$ and $w(\mathbf{i_2}) > \frac{\varepsilon}{2}$, the first enclosure is sufficient but the second is not. So the function calls itself recursively on $[m; v]$ with $depth - 1$ as the new maximal depth and $\varepsilon - w(\mathbf{i_1})$ as the new target accuracy, yielding $\mathbf{i_2'}$. The function then returns $\mathbf{i_1} + \mathbf{i_2'}$.
- If $w(\mathbf{i_1}) > \frac{\varepsilon}{2}$ and $w(\mathbf{i_2}) \leq \frac{\varepsilon}{2}$, we proceed symmetrically.
- Otherwise, the function calls itself on both $[u; m]$ and $[m; v]$ with $depth - 1$ as the new maximal depth and $\frac{\varepsilon}{2}$ as the new target accuracy, yielding $\mathbf{i_1'}$ and $\mathbf{i_2'}$. It then returns $\mathbf{i_1'} + \mathbf{i_2'}$.

This adaptive algorithm was chosen for its simplicity. One disadvantage is that it only has some local knowledge of how the integrand behaves. It would be interesting to compare it to more complicated algorithms, *e.g.* one that maintains a priority queue of all the subdomains and their associated integral so that it can split the subdomain with the widest integral overall [4].

4 Interval methods to approximate an improper integral

Improper integrals are computed by splitting the interval into two parts, a proper part which is treated with the previous methods, and the remainder which is handled in a specific way. The splitting is automatically performed by a variant of the adaptive method presented in Section 3.4 where the splitting point m for $[u; +\infty)$ is chosen to be $2u$ when $u > 0$.

In this section, we describe how we bound the remainder. We consider improper integrals of the shape $\int_u^v fg$ where either $u = 0^+$ or $v = +\infty$, and f is bounded. Function g belongs to a catalog of functions with known enclosures of their integral, such as $x^\alpha \ln^\beta x$. Section 4.1 presents the general theorem for integrals of the shape $\int_u^{+\infty} fg$, while Section 4.2 lists the functions g contained in our catalog. Finally, Section 4.3 focuses on integrals of the shape $\int_{0^+}^v fg$.

4.1 Improper integral of a product

To determine that $\int_u^{+\infty} h$ exists, we have added to Coquelicot a proof of the following Cauchy criterion: this integral exists if and only if for any $v \geq u$, $\int_u^v h$ exists and for all $\varepsilon > 0$, there exists $M > 0$ such that for all $u, v \geq M$, $|\int_u^v h| \leq \varepsilon$. We use this criterion to show the following lemma.

Lemma 4 *Let $f, g : \mathbb{R} \to \mathbb{R}$. Suppose that, on $[u; +\infty)$, f is bounded, f and g are continuous, and g has a constant sign. Moreover, suppose $\int_u^{+\infty} g$ exists. Then $\int_u^{+\infty} fg$ exists, and*

$$\int_u^{+\infty} fg \in \text{hull}\{f(t) \mid t \geq u\} \cdot \int_u^{+\infty} g.$$

Proof Since f is bounded on $[u; +\infty)$, let $[m; M] := \text{hull}\{f(t) \mid t \geq u\}$. Suppose without loss of generality that $g \geq 0$ on $[u; +\infty)$. Let $v \geq u$. For $u \leq t \leq v$, we have $m \cdot g(t) \leq f(t) \cdot g(t) \leq M \cdot g(t)$, hence $m \cdot \int_u^v g \leq \int_u^v fg \leq M \cdot \int_u^v g$. Let $\varepsilon > 0$. Since g is integrable, the Cauchy criterion gives some neighborhood P of $+\infty$ such that $\forall u, v \in P$, $|\int_u^v g| < \frac{\varepsilon}{1+\max(|m|,|M|)}$. But $|\int_u^v fg| \leq \max(|m|, |M|) \cdot \int_u^{+\infty} g < \varepsilon$; hence fg is integrable. Moreover $m \int_u^{+\infty} g \leq \int_u^{+\infty} fg \leq M \int_u^{+\infty} g$. Thus $\int_u^{+\infty} fg \in [m; M] \cdot \int_u^{+\infty} g$. If $g \leq 0$, the proof is similar.

We provide an effective version of the previous lemma, in the same spirit as Lemma 2, with a similar proof:

Lemma 5 *Let $F, I_g : \mathbb{I} \to \mathbb{I}$ be interval extensions respectively of f and $x \mapsto \int_x^{+\infty} g$. For any interval $\mathbf{u}$ such that $u \in \mathbf{u}$,*

$$\int_u^{+\infty} fg \in F(\text{hull}(\mathbf{u}, +\infty)) \cdot I_g(\mathbf{u}).$$

4.2 Catalog of supported integrable functions

In order to use Lemma 5, we need to be able to find a suitable extension I_g for the remainder of the integral of g. In that spirit, we look at two classes of well-known integrable functions.

4.2.1 Bertrand integrals

We consider functions $g(x) = x^\alpha \ln^\beta x$ with $\alpha \in \mathbb{R}, \beta \in \mathbb{R}$. These functions are of constant positive sign on $[1; +\infty)$. They are integrable at $+\infty$ only when $\alpha < -1$, or when $\alpha = -1$ and $\beta < -1$. Now we focus on how to compute them. If $\alpha < -1$, $\beta = 0$ and $u > 0$,

$$\int_u^{+\infty} x^\alpha \, dx = -\frac{u^{\alpha+1}}{\alpha + 1}. \tag{3}$$

When $\beta \geq 1$, integrating by parts shows that

$$\int_u^{+\infty} x^\alpha \ln^\beta x \, dx = -\left(\frac{u^{\alpha+1} \ln^\beta u}{\alpha+1}\right) - \frac{\beta}{\alpha+1} \int_u^{+\infty} x^\alpha \ln^{\beta-1} x \, dx. \tag{4}$$

Note that in order to prove this identity, we had to extend Coquelicot with a proof of the general formula for integration by parts.

When $\alpha < -1$ and $\beta < 0$, there is no closed form, but by moving $\ln^\beta x$ into the bounded part of Lemma 4, we can nevertheless compute bounds on the integral.

When $\alpha = -1$ and $\beta < -1$, we have a closed form:

$$\int_u^{+\infty} \frac{\ln^\beta x}{x} \, dx = -\frac{\ln^{\beta+1} u}{\beta+1}.$$

When $\alpha < -1$ and $\beta \geq 0$, and when moreover β is a natural number, we also have a closed form, obtained by recurrence on β using Equations (3) and (4). For instance, using (4) then (3), we get:

$$\int_1^{+\infty} \frac{\ln x}{x^2} \, dx = -\left(\frac{1^{-1} \ln 1}{-1}\right) - \frac{1}{-1} \int_1^{+\infty} \frac{dx}{x^2} = 0 + (-)\frac{1^{-1}}{-1} = 1.$$

4.2.2 Exponential

We also handle the case of the positive function $g(x) = e^{\gamma x}$ with $\gamma < 0$, using the fact that

$$\int_u^{+\infty} e^{\gamma x} \, dx = -\frac{e^{\gamma u}}{\gamma}.$$

4.3 Case of 0^+

When the singular bound is 0^+ instead of $+\infty$, we use a variant of Lemma 4.

Lemma 6 *Let $f, g : \mathbb{R} \to \mathbb{R}$. Suppose that, on $(0; v]$, f is bounded, f and g are continuous, and g has a constant sign. Moreover, suppose that $\int_{0^+}^v g$ exists. Then $\int_{0^+}^v fg$ exists, and*

$$\int_{0^+}^v fg \in \text{hull}\{f(t) \mid 0 \leq t \leq v\} \cdot \int_{0^+}^v g.$$

As in the case of $+\infty$, we have a catalog of supported functions. Consider $g(t) = t^\alpha (-\ln t)^\beta$ with $\alpha \in \mathbb{R}, \beta \in \mathbb{R}$. This function is of constant sign on $(0; v]$, where $v < 1$. Observe that using the substitution $t = \frac{1}{x}$, we get

$$\int_{0^+}^v t^\alpha (-\ln(t))^\beta dt = \int_{1/v}^\infty x^{-2-\alpha} \ln^\beta x \, dx.$$

The right-hand-side integral has the shape treated in Section 4.2.1, so we have a way to bound the left-hand-side integral. To do so, we added a proof of the substitution lemma to Coquelicot.

5 Automating the proof process

In this section we explain how to compute the approximations of the integrand (or of its bounded factor in the case of an improper integral) required by the theorems of Sections 3 and 4, and how to automate the proof of its integrability. We conclude by describing how all the ingredients combine into the implementation of a parameterized Coq tactic.

5.1 Straight-line programs and enclosures

As described in Section 2.3, enclosures and interval extensions are computed from expressions that appear as bounds or as the body of an integral, like for instance $\ln 2$, 3, and $(t+\pi)\sqrt{t}-(t+\pi)$, in $\int_{\ln 2}^{3}((t+\pi)\sqrt{t}-(t+\pi))\,dt$. The tactic represents these expressions symbolically, as straight-line programs. Such a program is a standard way of encoding directed acyclic graphs and thus of explicitly sharing common subexpressions. It is just a list of statements indicating what the operation is and where its inputs can be found. The place where the output is stored is left implicit: the result of an operation is always put at the top of the evaluation stack. Note that our evaluation model is simple: the stack grows linearly with the size of the expression since no element of the stack is ever removed. The stack is initially filled with values corresponding to the constants of the program. The result of evaluating a straight-line program is at the top of the stack.

Below is an example of a straight-line program corresponding to the expression $(t+\pi)\sqrt{t}-(t+\pi)$. It is a list containing the operations to be performed. Each list item first indicates the arity of the operation, then the operation itself, and finally the depth at which the inputs of the operation can be found in the evaluation stack. Note that, in this example, t and π are seen as constants, so the initial stack contains values that correspond to these subterms. The only thing that will later distinguish the integration variable t from an actual constant such as π is that the value of t is initially at the top of the evaluation stack. The comments in the term below indicate the content of the stack before evaluating each statement.

```
(* initial stack: [t, pi]                    *)     Binary Add 0 1
(* current stack: [t+pi, t, pi]              *)  :: Unary Sqrt 1
(* current stack: [sqrt t, t+pi, t, pi]      *)  :: Binary Mul 1 0
(* current stack: [(t+pi)*sqrt t, sqrt t, ...] *)  :: Binary Sub 0 2
(* current stack: [(t+pi)*sqrt t - (t+pi), ...] *)  :: nil
```

The evaluation of a straight-line program depends on the interpretation of the arithmetic operations and on the values stored in the initial stack. For instance, if the arithmetic operations are the operations from the `Reals` library (*e.g.* `Rplus`) and if the stack contains the symbolic value of the constants, then the result is the actual expression over real numbers.

Let us denote $[\![p]\!]_{\mathbb{R}}(\vec{x})$ the result of evaluating the straight-line program p with operators from `Reals` over an initial stack $\vec{x}$ of real numbers. Similarly, $[\![p]\!]_{\mathbb{I}}(\vec{\mathbf{x}})$ denotes the result of evaluating p with interval operations over a stack of intervals. Then, thanks to the inclusion property of interval arithmetic, we can prove the following formula once and for all:

$$\forall p,\ \forall \vec{x} \in \mathbb{R}^n,\ \forall \vec{\mathbf{x}} \in \mathbb{I}^n,\ (\forall i \le n,\ x_i \in \mathbf{x}_i) \Rightarrow [\![p]\!]_{\mathbb{R}}(\vec{x}) \in [\![p]\!]_{\mathbb{I}}(\vec{\mathbf{x}}). \qquad (5)$$

Formula (5) is the basic block used by the `interval` tactic for proving enclosures of expressions [11]. Given a goal $A \leq e \leq B$, the tactic first looks for a program p and a stack $\vec{x}$ of real numbers such that $[\![p]\!]_{\mathbb{R}}(\vec{x}) = e$. Note that this reification process is not proved to be correct, so Coq checks that both sides of the equality are convertible. More precisely, the goal $A \leq e \leq B$ is convertible to $[\![p]\!]_{\mathbb{R}}(\vec{x}) \in [A; B]$ if A and B are floating-point numbers and if the tactic successfully reified the term.

The tactic then looks in the context for hypotheses of the form $A_i \leq x_i \leq B_i$ so that it can build a stack $\vec{\mathbf{x}}$ of intervals such that $\forall i,\ x_i \in \mathbf{x}_i$. If there is no such hypothesis, the tactic just uses $(-\infty; +\infty)$ for $\mathbf{x}_i$. The tactic can now apply Formula (5) to replace the goal by $[\![p]\!]_{\mathbb{I}}(\vec{\mathbf{x}}) \subseteq [A; B]$. It then attempts to prove this new goal entirely by computation. Note that even if the original goal holds, this attempt may fail due to loss of correlation inherent to interval arithmetic.

Formula (5) also implies that if a function f can be reified as $t \mapsto [\![p]\!]_{\mathbb{R}}(t, \vec{x})$, then $\mathbf{t} \mapsto [\![p]\!]_{\mathbb{I}}(\mathbf{t}, \vec{\mathbf{x}})$ is an interval extension of f if $\forall i,\ x_i \in \mathbf{x}_i$. This way, we obtain the interval extensions of the integrand that we need for Sections 3 and 4.

There is also an evaluation scheme for computing RPAs for f. The program p is the same, but the initial evaluation stack now contains RPAs: a degree-1 polynomial for representing the domain of t, and constant polynomials for the constants. The result is an RPA of $t \mapsto [\![p]\!]_{\mathbb{R}}(t, \vec{x})$. By computing the image of this resulting polynomial approximation, one gets an enclosure of the expression that is usually better than the one computed by $\mathbf{t} \mapsto [\![p]\!]_{\mathbb{I}}(\mathbf{t}, \vec{\mathbf{x}})$.

5.2 Checking integrability

When computing the enclosure of an integral, the tactic should first obtain a formal proof that the integrand is integrable on the integration domain, as this is a prerequisite to all the theorems in Section 3. In fact we can be more clever by proving that, if we succeed in numerically computing an *informative* enclosure of the integral, the function was actually integrable. This way, the tactic does not have to prove anything beforehand about the integrand.

This trick requires to explain the inner workings of the CoqInterval library in more detail. In particular, the library provides evaluation schemes that use bottom values. In all that follows, $\overline{\mathbb{R}}$ denotes the set $\mathbb{R} \cup \{\bot_{\mathbb{R}}\}$ of *extended reals*, that is the set of real numbers completed with the extra point $\bot_{\mathbb{R}}$. The alternate scheme $[\![p]\!]_{\overline{\mathbb{R}}}$ produces the value $\bot_{\mathbb{R}}$ as soon as an operation is applied to inputs that are outside the usual definition domain of the operator. For instance, the result of dividing one by zero in $\overline{\mathbb{R}}$ is $\bot_{\mathbb{R}}$, while it is unspecified in $\mathbb{R}$. This $\bot_{\mathbb{R}}$ element is then propagated along the subsequent operations. Thus, the following equality holds, using the trivial embedding from $\mathbb{R}$ into $\overline{\mathbb{R}}$:

$$\forall p,\ \forall \vec{x} \in \mathbb{R}^n,\ [\![p]\!]_{\overline{\mathbb{R}}}(\vec{x}) \neq \bot_{\mathbb{R}} \Rightarrow [\![p]\!]_{\mathbb{R}}(\vec{x}) = [\![p]\!]_{\overline{\mathbb{R}}}(\vec{x}). \tag{6}$$

Moreover, the implementation of interval arithmetic uses not only pairs of floating-point numbers $[\inf \mathbf{x}; \sup \mathbf{x}]$ but also a special interval $\bot_{\mathbb{I}}$, which is propagated along computations. An interval operator produces the value $\bot_{\mathbb{I}}$ whenever the input intervals are not fully included in the definition domain of the corresponding real operator. In other words, an interval operator produces $\bot_{\mathbb{I}}$ whenever the corresponding operator on $\overline{\mathbb{R}}$ would have produced $\bot_{\mathbb{R}}$ for at least one value in one of the input intervals. Thus, by extending the definition of an enclosure so that $\bot_{\mathbb{R}} \in \bot_{\mathbb{I}}$ holds, we can prove a variant of Formula (5):

$$\forall p,\ \forall \vec{x} \in \overline{\mathbb{R}}^n,\ \forall \vec{\mathbf{x}} \in \mathbb{I}^n,\ (\forall i \le n,\ x_i \in \mathbf{x}_i) \Rightarrow [\![p]\!]_{\overline{\mathbb{R}}}(\vec{x}) \in [\![p]\!]_{\mathbb{I}}(\vec{\mathbf{x}}). \tag{7}$$

In CoqInterval, Formula (5) is actually just a consequence of both Formulas (6) and (7). This is due to two other properties of $\bot_{\mathbb{I}}$. First, $(-\infty; +\infty) \subseteq \bot_{\mathbb{I}}$ holds, so the conclusion of Formula (7) trivially holds whenever $[\![p]\!]_{\mathbb{I}}(\vec{\mathbf{x}})$ evaluates to $\bot_{\mathbb{I}}$. Second, $\bot_{\mathbb{I}}$ is the only interval containing $\bot_{\mathbb{R}}$. As a consequence, whenever $[\![p]\!]_{\mathbb{I}}(\vec{\mathbf{x}})$ does not evaluate to $\bot_{\mathbb{I}}$, the premise of Formula (6) holds.

Let us go back to the issue of proving integrability. By definition, whenever $[\![p]\!]_{\overline{\mathbb{R}}}(\vec{x})$ does not evaluate to $\bot_{\mathbb{R}}$, the inputs $\vec{x}$ are part of the definition domain of the expression represented by p. But we can actually prove a stronger property: not only is $\vec{x}$ part of the definition domain, it is also part of the continuity domain. More precisely, we can prove the following property:

$$\forall p,\ \forall t_0 \in \mathbb{R},\ \forall \vec{x} \in \mathbb{R}^n,\ [\![p]\!]_{\overline{\mathbb{R}}}(t_0, \vec{x}) \ne \bot_{\mathbb{R}} \Rightarrow$$
$$t \mapsto [\![p]\!]_{\mathbb{R}}(t, \vec{x}) \text{ is continuous at point } t_0. \tag{8}$$

Note that this property intrinsically depends on the operations that can appear inside p, *i.e.* the operations belonging to the class $\mathcal{E}$ of Section 2.3. Therefore, its proof has to be extended as soon as a new operator is supported in $\mathcal{E}$. In particular, it would become incorrect as such, if the integer part function was ever supported.

By combining Formulas (5) and (8), we obtain a numeric/symbolic method to prove that a function is continuous on a domain. Indeed, we just have to compute an enclosure of the function on that domain, and to check that it is not $\bot_{\mathbb{I}}$. A closer look at the way naive integral enclosures are computed provides the following corollary: whenever the enclosure of the integral is not $\bot_{\mathbb{I}}$, the function is actually continuous and thus integrable on any compact of the input domain. This solves the issue for proper integrals.

For improper integrals, the function has to be not only continuous but also bounded, *i.e.* its enclosure should have finite bounds in addition of being different from $\bot_{\mathbb{I}}$. This constraint incurs a usability issue in the case of an integration domain extending to $+\infty$. Indeed, the input domain $\vec{\mathbf{x}}$ is no longer bounded in that case, which means that RPAs become useless and one has to revert to a more naive interval evaluation. Let us illustrate the issue with the following integral for some lower bound $u > 0$:

$$\int_u^{+\infty} \frac{x+1}{x+2}\, e^{-x}\, dx.$$

The quotient is bounded on $[u; +\infty)$. Yet using naive interval arithmetic gives $[u+1; +\infty)/[u+2; +\infty) = [0; +\infty)$, which is not bounded. Thus the tactic is unable to prove integrability and to compute an enclosure of the integral. To circumvent this issue, the user has to massage the bounded part of the integrand into a form suitable for naive interval arithmetic, *e.g.* $1 - (x+2)^{-1}$. This time, the tactic obtains $[1 - (u+2)^{-1}; 1]$, which is bounded. However, this kind of transformation of the integrand is not always possible.

5.3 Integration into a tactic

The `interval` tactic is primarily dedicated to computing/verifying the enclosure of an expression. For this purpose, the expression is first turned into a straight-line program, as described in Section 5.1. There is however no integral operator in the grammar $\mathcal{E}$ of programs: from the point of view of the reification process, integrals are just constants, and thus part of the initial stack used when evaluating the program.

The tactic supports constants for which it can get a formally-proved enclosure. In previous releases of CoqInterval, the only supported constants were floating-point numbers and π. Floating-point numbers are enclosed by the corresponding point interval, which is trivially correct. An interval function and its correctness proof provide enclosures of the constant π, at the required precision.

The tactic now supports constants expressed as integrals $\int_u^v e\,dt$. First, it reifies the bounds u and v into programs and it evaluates them over $\mathbb{I}$ to get hopefully tight enclosures of them. In the case of an improper integral, only one of the bounds is reified; the other has to syntactically match either 0^+ or $+\infty$. Second, the tactic reifies e into a program p with t at the top of the initial evaluation stack. The tactic uses p to instantiate various evaluation methods, so that interval extensions and RPAs of e can be computed on all the integration subdomains, as described in Section 5.1. For improper integrals, the expression e has to be a product fg; the tactic then produces a program for f too, while g should syntactically match one of the functions of Section 4.2. Third, using the formulas of Sections 3 and 4, the tactic creates a term of type $\mathbb{I}$ that, once reduced by Coq's kernel, has actual floating-point bounds. The tactic also proves that this term is an enclosure of the integral, using the theorems of Sections 3, 4, and 5.2.

Regarding improper intervals, since the tactic only recognizes integrand of the form fg with g one of the functions of Section 4.2, it is up to the user to rewrite the integrand that way if it is not so already. Moreover, while g can in theory be of the shape $t^\alpha \ln^\beta t$, with α and β *arbitrary* exponents in the integrability range, the current implementation only supports *integer* exponents.

5.4 Controlling the tactic

The `interval` tactic features four options that supply the user with some control over how it computes integral enclosures. First, the user can indicate the target accuracy for the integral, expressed as a power-of-two upper bound on the width of the resulting enclosure. While an absolute bound is useful for benchmarks and in some degenerate cases, the user might prefer to specify a relative bound on the target accuracy. So another option makes it possible for the user to indicate how many bits of the result should be significant (by default, 10 bits, so about three decimal digits). It is an *a priori* bound, since the implementation first performs a coarse estimation of the integral value and uses it to turn the relative bound into an absolute one. It then performs computations using only this absolute bound.

The user can also indicate the degree of the RPAs used for approximating the integrand (default is 10). This value empirically provides a good compromise between bisecting too deeply and computing costly RPAs when targeting the default accuracy of 10 bits. For poorly approximated integrands, choosing a smaller degree can improve timings significantly, while for highly regular integrands and a high target accuracy, choosing a larger degree might be worth a try.

Finally, the user can limit the maximal depth of bisection (default is 3). If the target absolute error is reached on each interval of the subdivision, then increasing the maximal depth does not affect timings. There might, however, be some points of the integration domain around which the target error is never reached. This setting prevents the computations from splitting the domain indefinitely, while the computed enclosure is already accurate enough to prove the goal.

Note that as in previous CoqInterval releases, the user can adjust the precision of floating-point computations used for interval computations, which has an impact on how integrals are computed. The default value is 30 bits, which is sufficient in practice for getting the default 10 bits of integral accuracy.

There are three reasons why the user-specified target accuracy might not be reached. When specifying a relative bound, if the initial estimate of the integral is too coarse, the absolute bound used by the adaptive algorithm will be too large and the final result might be less accurate than desired. An insufficient bisection depth might also lead the result to be less accurate. This is also true with an insufficient precision of intermediate computations.

The following script shows how to prove in Coq that the surface of a quarter unit disk is equal to $\pi/4$, at least up to 10^{-6}. The target accuracy is set to 20 bits, so that we can hope to reach the 10^{-6} bound. Since the integrand is poorly approximated near 1 (due to the square root), the integration domain has to be split into small pieces around 1. So we significantly increase the bisection depth to 15. Finally, since the RPAs behave poorly here, decreasing their degree to 5 shaves a few tenths of second off the time needed to check the result. In the end, it takes under a second for Coq to formally check the proof.

```
Goal Rabs (RInt (fun t => sqrt (1 - t*t)) 0 1 - PI/4) <= 1/1000000.
interval with (i_integral_prec 20, i_integral_depth 15, i_integral_deg 5).
Qed.
```

6 Benchmarks

This section presents the behavior of the tactic on several integration problems, each given as a symbolic integral, its value (approximate if no closed form exists), and a set of absolute error bounds that must be reached by the tactic. Each problem is translated into a set of Coq scripts as follows, one for each bound:

```
Goal Rabs (RInt[_gen] function domain - value) <= error.
interval with options.
Qed.
```

The tactic options have been set using the following experimental protocol. The floating-point precision is set at about 10 more bits than the target accuracy, so that round-off errors do not worsen interval enclosures when summing integrals. The maximal depth is initially set to a large enough value. Then, various degrees of RPAs are tested and the one that leads to the fastest execution is kept. Finally, the maximal depth is reduced as long as the tactic succeeds in proving the bounds, so that we get an idea of how deep splitting has to be performed to compute an accurate enclosure of the integral. Note that reducing the maximal depth might improve timings in case the adaptive algorithm had been overly conservative and did too much domain splitting. Reducing the target accuracy could also improve timings (again by preventing some domain splitting), but this was not done.

The tables below indicate, for each error bound, the time needed and the tactic settings. Timings are in seconds and are obtained on a run-of-the-mill laptop from 2012 using Coq 8.7. All the timings below are obtained using the `vm_compute` machinery to perform computations. The tactic also supports the `native_compute` machinery [2], but its long startup time makes it useful only for the longest computations. So that the asymptotic complexity of our algorithms is more apparent, we chose to use only `vm_compute` in the benchmarks. But the reader should keep in mind that `native_compute` makes the tactic about twice as fast, *e.g.* the slowest benchmark below goes from 365 seconds down to 186 seconds.

6.1 Proper integrals

For each proper integral, we also ran several quadrature methods from Octave [5]: `quad`, `quadv`, `quadgk`, `quadl`, `quadcc`. We also used IntLab [16]; it provides `verify-quad`, an interval arithmetic procedure that computes integral enclosures using a verified Romberg method. For each method, we ask for an absolute accuracy of 10^{-15}. We only comment when the answer is off, or when the execution time exceeds 1 second. Finally, we also tested VNODE-LP [14] on each example by representing the integral as the value of the solution of a differential equation.

The first problem is the integral of the derivative of arctan, a highly regular function. As expected, the tactic behaves well on it, since it takes about 3 seconds to compute 18 decimal digits of π by integration. Note that the time needed for reifying the goal and performing the initial computations is incompressible, so there is not much difference between 10^{-3} and 10^{-6}.

$$\int_0^1 \frac{dx}{1+x^2} = \frac{\pi}{4}$$

Error	Time	Width	Deg	Depth	Prec
10^{-3}	0.2	2^{-10}	5	1	30
10^{-6}	0.2	2^{-20}	6	2	30
10^{-9}	0.4	2^{-30}	7	3	40
10^{-12}	0.7	2^{-40}	10	3	50
10^{-15}	1.0	2^{-50}	12	3	60
10^{-18}	1.7	2^{-60}	15	3	70

The second problem is Ahmed's integral [1]. It is a bit less regular and uses more operators than the previous problem, but the tactic still behaves well enough: adding ten bits of accuracy doubles the computation time.

$$\int_0^1 \frac{\arctan\sqrt{x^2+2}}{\sqrt{x^2+2}\,(x^2+1)}dx = \frac{5\pi^2}{96}$$

Error	Time	Width	Deg	Depth	Prec
10^{-3}	0.4	2^{-10}	3	2	30
10^{-6}	0.8	2^{-20}	7	2	30
10^{-9}	2.5	2^{-30}	11	2	40
10^{-12}	5.2	2^{-40}	11	3	50
10^{-15}	8.6	2^{-50}	13	3	60

The third problem involves a function that is harder to approximate using RPAs, so the tactic performs more domain splitting, degrading performances.

$$\int_0^\pi \frac{x \sin x}{1+\cos^2 x} dx = \frac{\pi^2}{4}$$

Error	Time	Width	Deg	Depth	Prec
10^{-3}	0.7	2^{-10}	5	3	30
10^{-6}	1.7	2^{-20}	7	4	30
10^{-9}	4.8	2^{-30}	11	4	40
10^{-12}	8.3	2^{-40}	13	4	50
10^{-15}	17.7	2^{-50}	17	4	60

The fourth problem is an example from Helfgott[3] in the spirit of [7]. The polynomial part crosses zero, so there is a point where the integrand is not differentiable because of the absolute value. Thus only degenerate Taylor models can be computed around that point. Although the tactic has to perform a lot of domain splitting to isolate that point, it still computes an enclosure of the integral quickly. Note that the approximate value of the integral was computed using the `interval_intro` tactic.

$$\int_0^1 \left| \left(x^4 + 10x^3 + 19x^2 - 6x - 6 \right) e^x \right| dx \simeq 11.14731055005714$$

On this example, quadrature methods have some troubles: `quad` gives only 10 correct digits; `verifyquad` gives a false answer (a tight interval not containing the value of the integral) without warning; `quadgk` gives only 9 correct digits. VNODE-LP cannot be used because of the absolute value. The bug of `verify_quad` lies in an incorrect implementation of Taylor models for absolute value; it has since then been fixed by removing support for absolute values.

Error	Time	Width	Degree	Depth	Precision
10^{-3}	0.6	2^{-10}	5	8	30
10^{-6}	0.9	2^{-20}	7	13	40
10^{-9}	1.3	2^{-30}	9	18	50
10^{-12}	1.9	2^{-40}	11	22	60
10^{-15}	2.6	2^{-50}	13	28	70

The last two problems are inherently hard to numerically integrate. The first one is the 12-th coefficient of a Chebyshev expansion. As with the previous problem, there are some points where no RPAs can be computed. The approximate value was again obtained using the `interval_intro` tactic.

$$\int_{-1}^1 \left(2048x^{12} - 6144x^{10} + 6912x^8 - 3584x^6 + 840x^4 - 72x^2 + 1\right) \\ \exp\left(-\left(x - \tfrac{3}{4}\right)^2\right) \sqrt{1-x^2}\, dx \simeq -3.2555895745 \cdot 10^{-6}$$

The `quad`, `quadl`, and `quadcc` procedures give completely off but consistent answers without warning; `quadv` gives an answer which is off the mark as well, but it gives a warning “maximum iteration count reached”; `verifyquad` works only for functions that are four times differentiable, hence its failure here; `quadgk` gives yet another off answer with no warning. Finally, VNODE-LP fails here because of computational errors such as divisions by 0.

[3] http://mathoverflow.net/questions/123677/rigorous-numerical-integration

Error	Time	Width	Degree	Depth	Precision
10^{-6}	10.0	2^{-20}	8	16	40
10^{-9}	21.0	2^{-30}	10	21	50
10^{-12}	43.1	2^{-40}	13	28	60
10^{-15}	94.2	2^{-50}	13	35	70

The last problem is an example taken from Tucker's book [17] and originally suggested by Rump [16, page 372]. This integral is often incorrectly approximated by computer algebra systems, because of the large number of oscillations (about 950 sign changes) and the large value of the n-th derivatives of the function. While the maximal depth is not too large, the tactic reaches it for numerous subdomains, hence the large computation time.

The `quad`, `quadcc`, and `quadgk` procedures give off values without any warning; `quadv` gives an off value with a warning; `verifyquad` takes 1.7 seconds to give a correct answer; `quadl` takes 9 seconds to return a correct answer.

$$\int_0^8 \sin(x + e^x)\, dx \simeq 0.3474$$

Error	Time	Width	Deg	Depth	Prec
10^{-1}	66.8	2^{-3}	6	12	30
10^{-2}	102.9	2^{-6}	5	13	30
10^{-3}	143.5	2^{-10}	6	13	30
10^{-4}	171.7	2^{-13}	7	13	30

6.2 Improper integrals

Few tools are able to handle unbounded integration domains and even fewer can give reliable bounds on the integral value. So this section is mostly about CoqInterval. The first example shows a simple integrand with an exponential bound:

$$\int_1^{+\infty} \frac{e^{-x}}{\sqrt{x}}\, dx = \sqrt{\pi} \cdot \operatorname{erfc}(1)$$

Error	Time	Width	Deg	Depth	Prec
10^{-3}	0.4	2^{-10}	7	3	30
10^{-6}	1.2	2^{-20}	7	5	30
10^{-9}	4.2	2^{-30}	9	7	40
10^{-12}	7.7	2^{-40}	13	7	50
10^{-15}	17.1	2^{-50}	13	8	60

The second example is similar to the integral from Tucker's book, in the sense that the oscillations of the integrand make it hard to accurately approximate the remainder. For instance, Maple 18 forfeits after 10 seconds of computations. The tactic does not perform much better since it is not able to compute more than two digits in a reasonable amount of time. This is partly due to the adaptive splitting algorithm, which is built upon the assumption that splitting an integration domain into two parts eventually improves the accuracy by more than one bit on each part; this is not the case for the remainder in this example.

$$\int_1^{+\infty} \cos x \frac{\ln x}{x^2}\, dx \simeq -0.1595$$

Error	Time	Width	Deg	Depth	Prec
10^{-1}	2.1	2^{-3}	12	10	30
10^{-2}	22.6	2^{-6}	14	16	30
10^{-3}	364.7	2^{-10}	19	23	30

The last example comes from Helfgott's proof of the ternary Goldbach conjecture [7, page 35]:

$$\int_{-\infty}^{\infty} \frac{(0.5 \cdot \ln(\tau^2 + 2.25) + 4.1396 + \ln \pi)^2}{0.25 + \tau^2}\, d\tau$$

The tactic cannot handle this integral fully automatically since the integrand is not syntactically a product with a term $x^\alpha \ln^\beta x$. It is up to the user to split the integral into two parts: one proper part between $-100{,}000$ and $100{,}000$ (as was done in the original paper) and one improper part between $100{,}000$ and $+\infty$ (counted twice, since the integrand is an even function). The proper part is handled in the same way as all the previous examples. It takes about 30 seconds to get the relative accuracy of 10^{-6} needed by the original paper. For the improper part, the integrand first has to be transformed into the following form, which was proved to be equal to the original integrand in a few lines of Coq:

$$\int_{100{,}000}^{+\infty} \frac{1 + \left(\frac{0.5\cdot\ln(1+2.25/\tau^2)+4.1396+\ln \pi}{\ln \tau}\right)^2}{1 + 0.25/\tau^2} \cdot \frac{\ln^2 \tau}{\tau^2}\, d\tau \simeq 3.17742 \cdot 10^{-3}.$$

Error	Time	Width	Degree	Depth	Precision
10^{-3}	1.1	2^{-10}	3	1	30
10^{-4}	1.6	2^{-13}	5	5	30
10^{-5}	2.8	2^{-16}	7	9	30
10^{-6}	5.5	2^{-20}	10	12	30
10^{-7}	9.8	2^{-23}	12	16	30
10^{-8}	17.6	2^{-26}	15	18	30

Bounding the remainder with a low accuracy is sufficient to prove that the integral on the whole domain is included in $[226.849; 226.850]$ and thus that the upper bound 226.844 used in [7] is incorrect.

7 Conclusion

We have presented a method for computing and formally verifying numerical enclosures of univariate definite integrals using the Coq proof assistant. This method has been integrated into the `interval` tactic. It provides formal proofs of the existence of integrals, in both proper and improper cases, and computes formally verified enclosures thereof. These proofs rely on the formal theory of Riemann integrals provided by our extension to the Coquelicot library. Note that our algorithms do not use anything specific to Riemann integrability and could be transposed to Lebesgue or gauge theories.

In the proper case, the enclosure method just requires that there exist rigorous polynomial approximations of the elementary functions in the integrand, so it is only limited by the underlying CoqInterval library. At the time of writing, the supported functions are $\sqrt{\cdot}$, cos, sin, tan, exp, ln, arctan, and the integer power function. Any new function added to the library would be supported almost immediately by the integration module.

The current treatment of improper integrals is less automated. In particular, the syntactic expression of the integrand has to make explicit the scale element that models its asymptotic behavior near the singularity. The tactic currently supports two scales: $e^{\gamma x}$ and $x^\alpha \ln^\beta x$. We could provide more scales to users, or at least merge these two into the more common scale $e^{\gamma x} x^\alpha \ln^\beta x$. More importantly, a more satisfactory tool for the improper case would require some support for the symbolic computation of expansions of the integrand along a given scale. This would both make the method more general and reduce the preparatory work required from the user.

Nested integrals are not supported by our method. The naive approach could easily be adapted to support them, but performances would be even worse due to the curse of dimensionality. As for the polynomial-based approach, it is not suitable for nested integrals, since there exists no general method for integrating multivariate polynomials. In fact, any 3-SAT instance can be reduced to approximating the integral of a multivariate polynomial.

While our adaptive bisection algorithm and our rigorous quadrature based on primitives of polynomial might seem crude, they proved effective in practice. They produce accurate approximations of non-pathological integrals in a few seconds, and thus they are usable in an interactive setting. Moreover, they can handle functions with unbounded second derivatives in a rigorous way, as well as unbounded integration domains. Another contribution of this paper is the way we are able to infer that a function is integrable from a successful computation of its integral.

For proper integrals, we could also have tried rigorous quadrature methods such as Newton-Cotes formulas. Rather than a degree-n approximation, the algorithm would integrate a degree-n polynomial interpolant of the integrand, which gives a much tighter enclosure of the integral at a fraction of the cost. The increased accuracy comes from the ability to compute a tight enclosure of the $n{+}1$-th derivative of the integrand. Unfortunately, CoqInterval only knows how to bound the first derivative. Note that a very simplified version of this approach has already been implemented in Coq in the setting of exact real arithmetic by O'Connor and Spitters [15]. Since it does not even involve the first derivative, it is akin to our naive approach and thus the performances are dreadful: computing $\int_0^1 \sin(x)\,dx$ up to three decimals takes 7 seconds. Comparatively, our tool computes 400 decimal digits in that same time, using degree-170 Taylor models and 1400 bits of precision. Note that such an accuracy is unattainable using Simpson's rule, even outside Coq, since it would require about 10^{99} point evaluations.

We could also have tried a much more general method, that is, solving a differential equation built from the integrand, as we did when using VNODE-LP. Again, there has been some work done for Coq in the setting of exact real arithmetic [10], but the performances are not good enough in practice. Much closer to actual numerical methods is Immler's work in Isabelle/HOL [8], which uses an arithmetic on affine forms. This approach is akin to computing with degree-1 RPAs.

Acknowledgements We would like to thank Érik Martin-Dorel for his improvements to the Coq framework for computing rigorous polynomial approximations and Philippe Dumas for stimulating discussions and suggestions.

References

1. Ahmed, Z.: Ahmed's integral: the maiden solution. Mathematical Spectrum **48**(1), 11–12 (2015)
2. Boespflug, M., Dénès, M., Grégoire, B.: Full reduction at full throttle. In: J.P. Jouannaud, Z. Shao (eds.) Certified Programs and Proofs, *LNCS*, vol. 7086, pp. 362–377. Springer, Kenting, Taiwan (2011). DOI 10.1007/978-3-642-25379-9_26
3. Boldo, S., Lelay, C., Melquiond, G.: Coquelicot: a user-friendly library of real analysis for Coq. Mathematics in Computer Science **9**(1), 41–62 (2015). DOI 10.1007/s11786-014-0181-1
4. Corliss, G.F., Rall, L.B.: Adaptive, self-validating numerical quadrature. SIAM Journal on Scientific and Statistical Computing **8**(5), 831–847 (1987). DOI 10.1137/0908069
5. Eaton, J.W., Bateman, D., Hauberg, S., Wehbring, R.: GNU Octave version 3.8.1 manual: a high-level interactive language for numerical computations (2014). URL http://www.gnu.org/software/octave/doc/interpreter
6. Hass, J., Schlafly, R.: Double bubbles minimize. Annals of Mathematics. Second Series **151**(2), 459–515 (2000). DOI 10.2307/121042
7. Helfgott, H.A.: Major arcs for Goldbach's problem (2014). URL http://arxiv.org/abs/1305.2897
8. Immler, F.: Formally verified computation of enclosures of solutions of ordinary differential equations. In: J.M. Badger, K.Y. Rozier (eds.) NASA Formal Methods (NFM), *LNCS*, vol. 8430, pp. 113–127. Springer (2014). DOI 10.1007/978-3-319-06200-6_9
9. Mahboubi, A., Melquiond, G., Sibut-Pinote, T.: Formally verified approximations of definite integrals. In: J.C. Blanchette, S. Merz (eds.) 7th Conference on Interactive Theorem Proving, *LNCS*, vol. 9807, pp. 274–289. Nancy, France (2016). DOI 10.1007/978-3-319-43144-4_17
10. Makarov, E., Spitters, B.: The Picard algorithm for ordinary differential equations in Coq. In: S. Blazy, C. Paulin-Mohring, D. Pichardie (eds.) 4th International Conference on Interactive Theorem Proving, *LNCS*, vol. 7998, pp. 463–468. Springer, Rennes, France (2013). DOI 10.1007/978-3-642-39634-2_34
11. Martin-Dorel, É., Melquiond, G.: Proving tight bounds on univariate expressions with elementary functions in Coq. Journal of Automated Reasoning pp. 1–31 (2015). DOI 10.1007/s10817-015-9350-4
12. Mayero, M.: Formalisation et automatisation de preuves en analyses réelle et numérique. Ph.D. thesis, Université Paris VI (2001)
13. Moore, R.E., Kearfott, R.B., Cloud, M.J.: Introduction to Interval Analysis. SIAM, Philadelphia, PA, USA (2009). DOI 10.1137/1.9780898717716
14. Nedialkov, N.S.: Interval tools for ODEs and DAEs. In: Scientific Computing, Computer Arithmetic and Validated Numerics (SCAN) (2006). DOI 10.1109/SCAN.2006.28. URL http://www.cas.mcmaster.ca/~nedialk/vnodelp/
15. O'Connor, R., Spitters, B.: A computer verified, monadic, functional implementation of the integral. Theoretical Computer Science **411**(37), 3386–3402 (2010)
16. Rump, S.M.: Verification methods: Rigorous results using floating-point arithmetic. Acta Numerica **19**, 287–449 (2010). DOI 10.1017/S096249291000005X. URL http://www.ti3.tu-harburg.de/rump/intlab/
17. Tucker, W.: Validated Numerics: A Short Introduction to Rigorous Computations. Princeton University Press, Princeton, NJ, USA (2011)