

2004

Formalizing O Notation in Isabelle / HOL

Jeremy Avigad

Carnegie Mellon University, avigad@cmu.edu

Kevin Donnelly

Carnegie Mellon University, kevind@cmu.edu

Follow this and additional works at: <http://repository.cmu.edu/philosophy>



Part of the [Philosophy Commons](#)

This Conference Proceeding is brought to you for free and open access by the Dietrich College of Humanities and Social Sciences at Research Showcase @ CMU. It has been accepted for inclusion in Department of Philosophy by an authorized administrator of Research Showcase @ CMU. For more information, please contact research-showcase@andrew.cmu.edu.

Formalizing O notation in Isabelle/HOL

Jeremy Avigad and Kevin Donnelly

Carnegie Mellon University

Abstract. We describe a formalization of asymptotic O notation using the Isabelle/HOL proof assistant.

1 Introduction

Asymptotic notions are used to characterize the approximate long-term behavior of functions in a number of branches of mathematics and computer science, including analysis, combinatorics, and computational complexity. Our goal here is to describe an implementation of one important asymptotic notion — “big O notation” — using the Isabelle/HOL proof assistant.

Developing a library to support such reasoning poses a number of interesting challenges. First of all, ordinary mathematical practice involving O notation relies on a number of *conventions*, some determinate and some ambiguous, so deciding on an appropriate formal representation requires some thought. Second, we will see that a natural way of handling the notation is inherently *higher-order*; thus the implementation is a way of putting the higher-order features of a theorem prover to the test. Finally, O notation is quite *general*, since many of the definitions and basic properties make sense for the analysis of any domain of functions $A \Rightarrow B$ where B has the structure of an ordered ring (or even, more generally, a ring with a valuation); in practice, A is often either the set of natural numbers or an interval of real numbers, and B may be \mathbb{N} , \mathbb{Q} , \mathbb{R} , or \mathbb{C} .

On the positive side, uses of O notation can have a very computational flavor, and making the right simplification rules available to an automated reasoner can yield effective results. Given the range of applications, then, this particular case study is a good test of the ability of today’s proof assistants to support an important type of mathematical analysis.

Section 2 provides a quick refresher course in O notation. Section 3 then reviews some of the specific features of Isabelle that are required for, and well-suited to, the task of formalization. In Sections 4 and 5 we describe our implementation. In Section 6 we describe our initial application, that is, deriving certain identities used in analytic number theory. Future work is described in Section 7. Piotr Rudnicki has drawn our attention to a similar formalization of asymptotic notions in Mizar [3, 4], which we also discuss in Section 7.

We are grateful to Clemens Ballarin for help with some of the type declarations in Section 4.

2 O notation

If f and g are functions, the notation $f(x) = O(g(x))$ is used to express the fact that f 's rate of growth is no bigger than that of g , in the following sense:

Definition 1. $f(x) = O(g(x))$ means $\exists c \forall x (|f(x)| \leq c \cdot |g(x)|)$.

Here it is assumed that f and g are functions with the same domain and codomain. The definition further assumes that notions of multiplication and absolute value are defined on the codomain; but otherwise the definition is very general.

Henceforth we will assume that the codomain is an ordered ring, giving us addition and subtraction as well. (We will also assume that rings are nondegenerate, i.e. satisfy $0 \neq 1$.) Absolute value is then defined by

$$|x| = \begin{cases} x & \text{if } 0 \leq x \\ -x & \text{otherwise} \end{cases}$$

This covers \mathbb{Z} , \mathbb{Q} , and \mathbb{R} . Here are some examples:

- As functions from the positive natural numbers, \mathbb{N}^+ , to \mathbb{Z} , $4x^2 + 3x + 12 = O(x^2)$.
- Similarly, $4x^2 + 3x + 12 = 4x^2 + O(x)$.
- If $f(n) = \sum_{k=0}^n \binom{3n}{k}$ is viewed as a function from \mathbb{N}^+ to \mathbb{Q} , we have

$$f(n) = \binom{3n}{n} \left(2 - \frac{4}{n} + O\left(\frac{1}{n^2}\right) \right).$$

This last example is taken from Graham et al. [1, Chapter 9], which is an excellent reference for asymptotic notions in general.

Some observations are in order. First, the notation in the examples belie the fact that O notation is about functions; thus, the more accurate rendering of the first example is

$$\lambda x. 4x^2 + 3x + 12 = O(\lambda x. x^2).$$

The corresponding shorthand is used all the time, even though it is ambiguous; for the example, the expression

$$ax^2 + bx + c = O(x^2)$$

is not generally true if one interprets the terms as functions of a , b , or c , instead of x .

The next thing to notice is that we have already stretched the notation well beyond Definition 1. For example, we read the second example above as the assertion that for some function f such that $f = O(x)$, $4x^2 + 3x + 12 = 4x^2 + f$.

The last thing to note is that in these expressions, “=” does not behave like equality. For example, although the assertion

$$x^2 + O(x) = O(x^3)$$

is correct, the symmetric assertion

$$O(x^3) = x^2 + O(x)$$

is not.

The approach described by Graham et al. [1] is to interpret $O(f)$ as denoting the *set* of all functions that have that rate of growth; that is,

$$O(f) = \{g \mid \exists c \forall x (|g(x)| \leq c \cdot |f(x)|)\}.$$

Then one reads equality in the expression $f = O(g)$ as the *element-of* relation, $f \in O(g)$. One can read the sum $g + O(h)$ as

$$g + O(h) = \{g\} + O(h) = \{g + k \mid k \in O(h)\},$$

and then interpret equality in the expression

$$g + O(h) = O(l)$$

as the subset relation, $g + O(h) \subseteq O(l)$. This is the reading we have followed.

The O operator can be extended to sets as well, by defining

$$O(S) = \bigcup_{f \in S} O(f) = \{g \mid \exists f \in S (f = O(g))\}.$$

Thus, if $f = g + O(h)$ and $h = k + O(l)$, we can make sense of the expression

$$f = g + O(k + O(l)).$$

There are various extensions to O notation, as we have described it. First of all, one often restricts the domain under consideration to some subset S of the domain of the relevant functions. For example, if f and g are functions with domain \mathbb{R} , we might say “ $f = O(g)$ on the interval $(0, 1)$,” thereby restricting the quantifier in the definition of $O(g)$. Second, one is often only concerned with the long-term behavior of functions, in which case one wants to ignore the behavior on any initial segment of the domain. The relation $f = O(g)$ is therefore often used instead to express the fact that f 's rate of growth is *eventually* dominated by that of g , i.e. that Definition 1 holds provided the universal quantifier is restricted to sufficiently large inputs. As an example, notice that the assertion $x + 1 \in O(x^2)$ is false on our strict reading if x is assumed to range over the natural numbers, since no constant satisfies the definition for $x = 0$. The statement *is* true, however, on the interval $[1, \infty)$, as well as eventually true.

There is an elegant way of modeling these variants in our framework. If A is any set of functions from σ to τ and S is any set of elements of type σ , define “ A on S ” to be the set

$$\{f \mid \exists g \in A \forall x \in S (f(x) = g(x))\}$$

of all functions that agree with some function in A on elements of S . The second variant assumes that there is an ordering on the domain, but when this is the case, we can define “ A eventually” to be the set

$$\{f \mid \exists k \exists g \in A \forall x \geq k (f(x) = g(x))\}$$

of all functions that eventually agree with some function in A . With these definitions, “ $f = O(g)$ on S ” and “ $f = O(g)$ eventually” have the desired meanings.

Finally, as noted in the introduction, O notation makes sense for any ring with a suitable *valuation*, such as the modulus function $|\cdot| : \mathbb{C} \Rightarrow \mathbb{R}$ on the complex numbers. Such a variation would require only minor changes to the definitions and lemmas we give below; conversely, if we define $O'(f) = O(\lambda x. |f(x)|)$, properties of the more general version can be derived from properties of the more restricted one. Below, we will only treat the most basic version of O notation, described above.

3 Isabelle preliminaries

Isabelle [12] is a generic proof assistant developed under the direction of Larry Paulson at Cambridge University and Tobias Nipkow at TU Munich. The HOL instantiation [6] provides a formal framework based on Church’s simple type theory. In addition to basic types like *nat* and *bool* and type constructors for product types, function types, set types, and list types, one can turn any set of objects into a new type using a **typedef** declaration. Isabelle also supports polymorphic types and overloading. Thus, if $'a$ is a variable ranging over types, then in the term $(x::'a) + y$ it is inferred that y has type $'a$, and that $'a$ ranges over types for which an operation $+$ has been defined.

Moreover, in Isabelle/HOL types are grouped together into *sorts*. These sorts are ordered, and types of a sort inherit properties of any of the sort’s ancestors. Thus the sorts, also known as *type classes*, form a hierarchy, with the predefined *logic* class, containing all types, at the top. This, in particular, supports subtype polymorphism; for example, the term

$$(\lambda x y. x \leq y)::('a::order) \Rightarrow 'a \Rightarrow bool$$

represents an object of type $'a \Rightarrow 'a \Rightarrow bool$, where $'a$ ranges over types of the sort *order*. Here the notation $t::(T::S)$ means that the term t has type T , which is a member of sort S .

Isabelle also allows one to impose axiomatic requirements on a sort; that is, an *axiomatic type class* is simply a class of types that satisfy certain axioms. Axiomatic type classes make it possible to introduce terms, concepts, and associated theorems at a useful level of generality. For example, the classes

```
axclass plus < term
axclass zero < term
axclass one < term
```

are used to define the classes of types, after which polymorphic constants

```

+::('a::plus) ⇒ 'a ⇒ 'a
0::('a::zero)
1::('a::one)

```

can be declared to exist simultaneously for all types $'a$ in the respective classes. The following declares the class $plus-ac0$ to be a class of types for which 0 and + are defined and satisfy appropriate axioms:

```

axclass plus-ac0 < plus, zero
  commute: x + y = y + x
  assoc: (x + y) + z = x + (y + z)
  zero: 0 + x = x

```

One can then derive theorems that hold for any type in this class, such as

```

theorem right-zero: x + 0 = x::('a::plus-ac0)

```

One can also define operations that make sense for any member of a class. For example, the Isabelle HOL library defines a general summation operator

```

setsum::('a ⇒ 'b) ⇒ (('a set) ⇒ ('b::plus-ac0))

```

Assuming $'b$ is any type in the class $plus-ac0$, A is any set of objects of type $'a$, and f is any function from $'a$ to $'b$, $setsum A f$ denotes the sum $\sum_{x \in A} f(x)$. (This operator is defined to return 0 if A is infinite.) The assertion

```

instance nat::plus-ac0

```

declares the particular type nat to be an instance of $plus-ac0$, and requires us to prove that elements of nat satisfy the defining axioms. Once this is done, however, we can use operators like $setsum$ and the general results proved for $plus-ac0$ freely for nat .

More details on Isabelle's mechanisms for handling axiomatic type classes and overloading can be found in [8, 9].

4 The formalization

Our formalization makes use of Isabelle 2003's HOL-Complex library. This includes a theory of the real numbers and the rudiments of real analysis (including nonstandard analysis), developed by Jacques Fleuriot (with contributions by Larry Paulson). It also includes an axiomatic development of rings by Markus Wenzel and Gertrud Bauer, which was important to our formalization.

Aiming for generality, we designed our library to deal with functions from any set into a nondegenerate ordered ring. O sets are defined as described by Definition 1 in Section 2:

$$O(g) = \{h \mid \exists c \forall x (|h(x)| \leq c \cdot |g(x)|)\}$$

When f and g are elements of a function type such that addition is defined on the codomain, we would like define $f + g$ to be their pointwise sum. Similarly, if

A and B are sets of elements of a type for which addition is defined, we would like to define the sum $A + B$ to be the set of sums of their elements. The first step is to declare the appropriate function and set types to be appropriate for such overloading:

```
instance set :: (plus)plus
instance fun :: (type,plus)plus
```

We may then define the corresponding operations:

```
defs (overloaded)
  func-plus: f + g == (λx. f x + g x)
  set-plus: A + B == {c. ∃ a∈A. ∃ b∈B. c = a + b}
```

We can define multiplication and subtraction similarly, assuming these operations are supported by the relevant types. We can also lift a zero element:

```
instance fun :: (type,zero)zero
instance set :: (zero)zero
defs (overloaded)
  func-zero: 0::('a::type) ⇒ ('b::zero) == λx. 0
  set-zero: 0::('a::zero)set == {0}
```

In other words, the 0 function is the constant function which returns 0, and the 0 set is the singleton containing 0. Similarly, one can define appropriate notions of 1.

Now, assuming the types in question are elements of the sort *plus-ac0*, we can show that the liftings to the function and set types are again elements of *plus-ac0*.

```
instance fun :: (type,plus-ac0)plus-ac0
instance set :: (plus-ac0)plus-ac0
```

This declaration requires justification: we have to show that the resulting addition operation is associative and commutative, and that the zero element is an additive identity. Similarly, if the relevant starting type is a ring, then the resulting function type is a ring:

```
instance fun :: (type,ring)ring
```

Similarly, if the underlying type is a ring, multiplication of sets is commutative and associative, distributes over addition, and has an identity, 1.

We can now define the operation O , which maps a suitable function f to the set of functions, $O(f)$:

```
constdefs
  bigo :: ('a ⇒ 'b::oring-nd) ⇒ ('a ⇒ 'b) set      ((IO'(-)))
  O(f::('a ⇒ 'b::oring-nd)) == {h. ∃ c. ∀ x. abs (h x) ≤ c * abs (f x)}
  bigoset :: ('a ⇒ 'b::oring-nd) set ⇒ ('a ⇒ 'b) set  ((IO'(-)))
  O(S::('a ⇒ 'b::oring-nd) set) == {h. ∃ f∈S. ∃ c. ∀ x.
    abs(h x) ≤ (c * abs(f x))}
```

Recall that when it comes to O notation we also want to be able to deal with terms of the form $f + O(g)$, where f and g are functions. Thus we define an operation “+ o ” that takes, as argument, an element of a type and a set of elements of the same type:

```
constdefs
  elt-set-plus :: 'a::plus  $\Rightarrow$  'a set  $\Rightarrow$  'a set (infixl +o 70)
  a +o B == {c.  $\exists b \in B. c = a + b$ }
```

The operation $*o$ is defined similarly. The following declaration indicates that these operations should be displayed as “+” and “*” in all forms of output from the theorem prover:

```
syntax (output)
  elt-set-plus :: 'a  $\Rightarrow$  'a set  $\Rightarrow$  'a set (infix + 70)
  elt-set-times :: 'a  $\Rightarrow$  'a set  $\Rightarrow$  'a set (infix * 80)
```

Remember that according to the conventions described in Section 2, we would like to interpret $x = O(f)$ as $x \in O(f)$, and, for example, $f + O(g) = O(h)$ as $f + O(g) \subseteq O(h)$. Thus we declare symbols “= o ” and “= s ” to denote these two “equalities”:

```
syntax
  elt-set-eq :: 'a  $\Rightarrow$  'a set  $\Rightarrow$  bool (infix =o 50)
  set-set-eq :: 'a set  $\Rightarrow$  'a set  $\Rightarrow$  bool (infix =s 50)
translations
  x =o A => x  $\in$  A
  A =s B => A  $\subseteq$  B
```

Because they are translated immediately, the symbols = o and = s are displayed as \in and \subseteq , respectively, in the prover’s output. In a slightly underhanded way, however, we can arrange to have = o , = s , + o , and $*o$ appear as =, =, +, and *, respectively, in proof scripts: we introduce new symbols to denote the former, and then configure our editor, document generator, etc. to display these symbols as the latter.

The advantage to having these translations in place is that our formalizations come closer to textbook expressions. The following table gives some examples of ordinary uses of O notation, paired with our Isabelle representations:

$f = g + O(h)$	$f = g + O(h)$
$x^2 + 3x + 1 = x^2 + O(x)$	$(\lambda x. x^2 + 3 * x + 1) = (\lambda x. x^2) + O(\lambda x. x)$
$x^2 + O(x) = O(x^2)$	$(\lambda x. x^2) + O(\lambda x. x) = O(\lambda x. x^2)$

The equality symbol should be used to denote \in and \subseteq , however, only when the context makes this usage clear. For example, it is ambiguous as to whether the expression $O(f) = O(g)$ refers to set equality or the subset relation, that is, whether the equality symbol is the ordinary one or a translation of the equality symbol we have defined for O notation. For that reason, we will use \in and \subseteq when describing properties of the notation in Section 5. However, we resort to the common textbook uses of equality when we describe some particular identities in Section 6.

5 Basic properties

In this section we indicate some of the theorems in the library that we have developed to support O notation. As will become clear in Section 6, a substantial part of reasoning with the notation involves reasoning about relationships between sets with respect to addition of sets, and addition of elements and sets. Here are some of the basic properties:

<i>set-plus-intro</i>	$[a \in C, b \in D] \Rightarrow a + b \in C + D$
<i>set-plus-intro2</i>	$b \in C \Rightarrow a + b \in a + C$
<i>set-plus-rearrange</i>	$(a + C) + (b + D) = (a + b) + (C + D)$
<i>set-plus-rearrange2</i>	$a + (b + C) = (a + b) + C$
<i>set-plus-rearrange3</i>	$(a + C) + D = a + (C + D)$
<i>set-plus-rearrange4</i>	$C + (a + D) = a + (C + D)$
<i>set-zero-plus</i>	$0 + C = C$

Here a and b range of elements of a type of sort *plus-ac0*, C and D range over arbitrary sets of elements of such a type, and in an expression like $a + C$, the symbol $+$ really denotes the $+o$ operator. The bracket notation in *set-plus-intro* means that the conclusion $a + b \in C + D$ follows from the two hypotheses $a \in C$ and $b \in D$. If we use the four rearrangments above to simplify a term built up from the three types of addition, the result is a term consisting of a sum of elements on the left “added” to a sum of sets on the right. This makes it easy verify identities that enable one to rearrange terms in a calculation.

Since reasoning about expressions involving O notation essentially boils down to reasoning about inclusions between the associated sets, the following monotonicity properties are central:

<i>set-plus-mono</i>	$C \subseteq D \Rightarrow a + C \subseteq a + D$
<i>set-plus-mono2</i>	$[C \subseteq D, E \subseteq F] \Rightarrow C + E \subseteq D + F$
<i>set-plus-mono3</i>	$a \in C \Rightarrow a + D \subseteq C + D$
<i>set-plus-mono4</i>	$a \in C \Rightarrow a + D \subseteq D + C$

These are declared to the automated reasoner. We will see in Section 6 that, in conjunction with the properties below, this provides useful support for asymptotic calculations.

Analogous lists of properties holds for set multiplication, under the assumption that the multiplication on the underlying type is commutative and associative, with an identity. Assuming the underlying type is a ring, the distributivity of multiplication over addition lifts in various ways:

<i>set-times-plus-distrib</i>	$a * (b + C) = a * b + a * C$
<i>set-times-plus-distrib2</i>	$a * (C + D) = a * C + a * D$
<i>set-times-plus-distrib3</i>	$(a + C) * D \subseteq a * D + C * D$

The following theorem relates ordinary subtraction to element-set addition:

<i>set-minus-plus</i>	$(a - b \in C) = (a \in b + C)$
-----------------------	---------------------------------

Note that equality here denotes propositional equivalence.

Turning now to O notation proper, the following properties follow more or less from the basic set-theoretic properties of the definitions:

<i>bigo-elt-subset</i>	$f \in O(g) \Rightarrow O(f) \subseteq O(g)$
<i>bigoset-elt-subset</i>	$f \in O(A) \Rightarrow O(f) \subseteq O(A)$
<i>bigoset-mono</i>	$A \subseteq B \Rightarrow O(A) \subseteq O(B)$
<i>bigo-refl</i>	$f \in O(f)$
<i>bigoset-refl</i>	$A \subseteq O(A)$
<i>bigo-bigo-eq</i>	$O(O(f)) = O(f)$

Here, variables like f and g range over functions whose codomain is a nondegenerate ordered ring, and A and B range over sets of such functions.

In the definition of $O(f)$, we can assume without loss of generality that c is strictly positive. (Here we assume $0 \neq 1$.) It is easier not to have to show this when demonstrating that the definitions are met; on the other hand, the next three theorems allow us to use this fact where convenient.

<i>bigo-pos-const</i>	$(\exists c \forall x (h(x) \leq c * f(x))) =$ $(\exists c (0 < c \wedge \forall x (h(x) \leq c * f(x))))$
<i>bigo-alt-def</i>	$O(f) = \{h \mid \exists c (0 < c \wedge \forall x (h(x) \leq c * g(x))\}$
<i>bigoset-alt-def</i>	$O(A) = \{h \mid \exists f \in A \exists c (0 < c \wedge \forall x$ $(h(x) \leq c * g(x))\}$

The following is an alternative characterization of the O operator applied to sets.

<i>bigoset-alt-def2</i>	$O(A) = \{g \mid \exists f \in A (h \in O(f))\}$
-------------------------	--

The following properties are useful for calculations. Expressed at this level of generality, their proofs can only rely on properties, like the triangle inequality, that are true in every nondegenerate ordered ring.

<i>bigo-plus-idemp</i>	$O(f) + O(f) = O(f)$
<i>bigo-plus-subset</i>	$O(f + g) \subseteq O(f) + O(g)$
<i>bigo-plus-subset2</i>	$O(f + A) \subseteq O(f) + O(A)$
<i>bigo-plus-subset3</i>	$O(A + B) \subseteq O(A) + O(B)$
<i>bigo-plus-subset4</i>	$[[\forall x (0 \leq f(x)), \forall x (0 \leq g(x))]] \Rightarrow$ $O(f + g) = O(f) + O(g)$
<i>bigo-plus-absorb</i>	$f \in O(g) \Rightarrow f + O(g) = O(g)$
<i>bigo-plus-absorb2</i>	$[f \in O(g), A \subseteq O(g)] \Rightarrow f + A \subseteq O(g)$

To see that the subset relation cannot be replaced by equality in *bigoplus-subset*, consider what happens when $g = -f$. For most calculations, the subset relation is sufficient; but when the relevant functions are positive, *bigoplus-subset4* can simplify matters.

The following group of theorems is also useful for calculations.

<i>bigoplus-mult</i>	$O(f) * O(g) \subseteq O(f * g)$
<i>bigoplus-mult2</i>	$f * O(g) \subseteq O(f * g)$
<i>bigoplus-minus</i>	$f \in O(g) \Rightarrow -f \in O(g)$
<i>bigoplus-minus2</i>	$f \in g + O(h) \Rightarrow -f \in -g + O(h)$
<i>bigoplus-minus3</i>	$O(-f) = O(f)$
<i>bigoplus-add-commute</i>	$(f \in g + O(h)) = (g \in f + O(h))$

Showing that a particular function is in a particular O set is often just a matter of unwinding definitions. The following theorems offer shortcuts:

<i>bigoplus-bounded</i>	$[\forall x (0 \leq f(x)), \forall x (f(x) \leq g(x))] \Rightarrow f \in O(g)$
<i>bigoplus-bounded2</i>	$[\forall x (g(x) \leq f(x)), \forall x (f(x) \leq g(x) + h(x))] \Rightarrow f \in g + O(h)$

The next two theorems only hold for ordered rings with the additional property that for every nonzero c there is a d such that $cd \geq 1$. They are therefore proved under this hypothesis, which holds for any field, as well as for any archimedean ring.

<i>bigoplus-const</i>	$c \neq 0 \Rightarrow O(\lambda x.c) = O(1)$
<i>bigoplus-const-mult</i>	$c \neq 0 \Rightarrow O((\lambda x.c) * f) = O(f)$
<i>bigoplus-const-mult2</i>	$c \neq 0 \Rightarrow (\lambda x.c) * O(f) = O(f)$

In all three cases, the \subseteq direction holds more generally for ordered rings, without the requirement that $c \neq 0$.

Additional properties of O notation can be shown to hold in more specialized situations. For example, the HOL-Complex library includes a function *sumr m n f*, intended to denote $\sum_{m \leq i < n} f(i)$, where m and n are elements of \mathbb{N} and f is a function from \mathbb{N} to \mathbb{R} . Using the more perspicuous notation for sums, we have the following theorems:

<i>bigoplus-sumr-pos</i>	$[\forall x (0 \leq h(x)), f \in O(h)] \Rightarrow$ $\lambda x. \sum_{i < x} f(i) \in O(\lambda x. \sum_{i < x} h(i))$
<i>bigoplus-sumr-pos2</i>	$[\forall x (0 \leq h(x)), f \in g + O(h)] \Rightarrow$ $\lambda x. \sum_{i < x} f(i) \in \lambda x. \sum_{i < x} g(i) + O(\lambda x. \sum_{i < x} h(i))$

6 Application: arithmetic functions

In this section, we will describe an initial application of our library. All of the following facts are used in analytic number theory:

$$\ln(1 + 1/(n + 1)) = 1/(n + 1) + O(1/(n + 1)^2) \quad (1)$$

$$\sum_{i < n+1} \frac{1}{i + 1} = \ln(n + 1) + O(1) \quad (2)$$

$$\sum_{i < n+1} \ln(i + 1) = (n + 1) \ln(n + 1) - n + O(\ln(n + 1)) \quad (3)$$

$$\sum_{i < n} \frac{\ln(i + 1)}{i + 1} = \frac{1}{2} \ln^2(n + 1) + O(1) \quad (4)$$

$$\ln^2(n + 2) - \ln^2(n + 1) = 2 \frac{\ln(n + 1)}{n + 1} + O\left(\frac{\ln(n + 1) + 1}{(n + 1)^2}\right) \quad (5)$$

$$\sum_{i < n+1} \ln^2(i + 1) = \frac{(n + 1) \ln^2(n + 1) - 2(n + 1) \ln(n + 1) + 2n + O(\ln^2(n + 1))}{2} \quad (6)$$

$$\sum_{i < n+1} \ln^2\left(\frac{n + 1}{i + 1}\right) = 2n + O(\ln^2(n + 1)) \quad (7)$$

In these identities, the relevant terms are to be viewed as functions $f(n)$ from the natural numbers \mathbb{N} to the real numbers \mathbb{R} , and all the sums range over natural numbers. Keep in mind that here “equality” really means “element of”!

A more natural formulation of the first identity, for example, would be

$$\ln(1 + 1/n) = 1/n + O(1/n^2) \quad \text{for } n \geq 1,$$

and a more natural formulation of the second identity would be

$$\sum_{i=1}^n 1/i = \ln(n) + O(1) \quad \text{for } n \geq 1.$$

We have found it convenient to replace n by $n + 1$ instead of dealing with the side condition or using the type of positive natural numbers instead.

All of the identities above have been formalized in Isabelle. Our formalizations of the first three look as follows:

$$(\lambda n::nat. \ln (1 + 1 / (\text{real } n + 1))) = (\lambda n. 1 / (\text{real } n + 1)) + O(\lambda n. 1 / ((\text{real } n) + 1) ^2)$$

$$(\lambda n. \text{sumr } 0 (n+1) (\lambda i. 1/(\text{real } i + 1))) = (\lambda n. \ln(\text{real } n + 1)) + O(\lambda n. 1)$$

$$\begin{aligned} (\lambda n. \text{sumr } 0 (n+1) (\lambda i. \ln(\text{real } i + 1))) = \\ ((\lambda n. (\text{real } n + 1) * \ln(\text{real } n + 1)) - (\lambda n. \text{real } n)) + \\ O(\lambda n. \ln (\text{real } n + 1)) \end{aligned}$$

These are reasonable approximations to the usual mathematical notation, but for readability, we will use the latter below.

In an ordinary mathematical development, the first identity has to be proved, somehow, from the definition of \ln . (It can be seen immediately, for example, from the Taylor series expansion of $\ln(1+x)$ at $x=0$.) The others are typically derived from this using direct calculation, as well as, sometimes, basic properties of integration; see, for example, [5, 7]. For example, the second identity reflects the fact that $\ln x$ is equal to $\int_1^x (1/y)dy$ and the third reflects the identity $\int \ln x = x \ln x + C$, which may be obtained using integration by parts.

Since Isabelle’s current theory of integration was not always robust enough to handle the standard arguments, we had to replace these arguments by more direct proofs. This was an illuminating exercising in “unwinding” the methods of calculus. Thus, (5) above was useful in our low-tech proofs of the (4) and (6). Even *with* calculus, however, a good deal of ordinary asymptotic calculations are involved. To illustrate the use of our library, we will show how the theorems described in Section 5 are used to obtain (6) from (2), (3), (4), and (5).

To prove (6), first note that it suffices to establish the slightly weaker identity with $O(\ln^2(n+1)+1)$ in place of $O(\ln^2(n+1))$. The stronger version can then be obtained by unwinding definitions, using the fact that the terms on each side of the equation are equal when $n=0$, and $\ln^2(n+1) \geq \ln^2 2$ when $n > 0$.

First of all, using the technique of partial summation [5, 7], we have

$$\sum_{i < n+1} \ln^2(i+1) = (n+1) \ln^2(n+1) - \sum_{i < n} (i+1)(\ln^2(i+2) - \ln^2(i+1)). \quad (8)$$

This identity can be verified directly using induction on n . To estimate the second term on the right side, we start by multiplying (5) through by $n+1$, and using *set-times-plus-distrib* and *big-mult2* to obtain

$$(n+1)(\ln^2(n+2) - \ln^2(n+1)) = 2 \ln(n+1) + O\left(\frac{\ln(n+1)+1}{n+1}\right).$$

Using *big-sumr-pos2*, we obtain

$$\begin{aligned} \sum_{i < n} (i+1)(\ln^2(i+2) - \ln^2(i+1)) &= 2 \sum_{i < n} \ln(i+1) + O\left(\sum_{i < n} \frac{\ln(i+1)+1}{i+1}\right) \\ &= 2 \sum_{i < n} \ln(i+1) + O\left(\sum_{i < n} \frac{\ln(i+1)}{i+1}\right) + O\left(\sum_{i < n} \frac{1}{i+1}\right) \end{aligned} \quad (9)$$

Here, the second “equality” is really set inclusion, and is obtained using properties of sums, *big-plus-subset*, and *set-plus-mono*.

Now, let us estimate each of the three terms on the right-hand side, keeping in mind that we only care about equality up to $O(\ln^2(n+1)+1)$. Considering

the first term, we have

$$\begin{aligned}
\sum_{i < n} \ln(i + 1) &= \sum_{i < n+1} \ln(i + 1) - \ln(n + 1) \\
&= -\ln(n + 1) + ((n + 1) \ln(n + 1) - n + O(\ln(n + 1))) \\
&= ((n + 1) \ln(n + 1) - n) + (-\ln(n + 1) + O(\ln(n + 1))) \\
&= (n + 1) \ln(n + 1) - n + O(\ln^2(n + 1) + 1).
\end{aligned}$$

The second equality is obtained by applying *set-plus-intro2* to identity (3). In the third equality, we use *set-plus-rearrange2* and ordinary properties of real addition to group the terms appropriately. The last equality uses the fact that $\ln(n + 1)$ is in $O(\ln^2(n + 1) + 1)$, and so, by *big-minus-eq*, $-\ln(n + 1)$ is in $O(\ln^2(n + 1) + 1)$ as well; using *big-elt-subset* and *big-plus-absorb*, the second parenthetical expression is a subset of $O(\ln^2(n + 1) + 1)$, and the equality follows from *set-plus-mono*. Calculations like these work well with Isabelle/Isar's support for calculational reasoning [10], since intermediate identities can often be verified automatically. For example, the second-to-last equality above is verified by simplifying terms; the last equality is also confirmed by the automated reasoner, given only the fact that $\ln(n + 1)$ is in $O(\ln^2(n + 1) + 1)$. Multiplying through by 2, and using *set-times-plus-distrib* and *big-const-mult2*, we obtain

$$2 \sum_{i < n} \ln(i + 1) = (2(n + 1) \ln(n + 1) - 2n) + O(\ln^2(n + 1) + 1).$$

Turning to the second term, we have

$$\begin{aligned}
O\left(\sum_{i < n} \frac{\ln(i + 1)}{i + 1}\right) &= O(\ln^2(n + 1)/2 + O(1)) \\
&= O(\ln^2(n + 1)/2) + O(1) \\
&= O(\ln^2(n + 1) + 1) + O(\ln^2(n + 1) + 1) \\
&= O(\ln^2(n + 1) + 1).
\end{aligned}$$

The first equality is obtained by applying *big-elt-subset* to identity (4). The second equality uses *big-plus-subset3* and *big-bigo-eq*. Finally, we use *big-elt-subset*, monotonicity properties of set addition, and *big-plus-idemp*, as before, to simplify the resulting O set. (Alternatively, one can use (4) to show $\sum_{i < n} \ln(i + 1)/(i + 1) = O(\ln^2(n + 1) + 1)$, and then use *big-elt-subset*.)

Turning to the third term, we have

$$\begin{aligned}
O\left(\sum_{i < n} \frac{1}{i + 1}\right) &= O\left(\sum_{i < n+1} \frac{1}{i + 1} + (-1/(n + 1))\right) \\
&= O(\ln(n + 1) + O(1)) + O(1/(n + 1)) \\
&= O(\ln(n + 1)) + O(1) + O(1) \\
&= O(\ln^2(n + 1) + 1)
\end{aligned}$$

using reasoning similar to that above.

Returning to equation (9), grouping terms appropriately, and using monotonicity of addition, we obtain

$$\sum_{i < n} (i + 1)(\ln^2(i + 2) - \ln^2(i + 1)) = 2(n + 1) \ln(n + 1) - 2n + O(\ln^2(n + 1) + 1).$$

Using *bigo-minus2* to negate both sides, substituting the result into (8), and applying *set-plus-intro2*, we have

$$\sum_{i < n+1} \ln^2(i + 1) = (n + 1) \ln^2(n + 1) - 2(n + 1) \ln(n + 1) + 2n + O(\ln^2(n + 1) + 1),$$

as required.

7 Future work

The library described here is being used to obtain a fully formalized proof of the prime number theorem, which states that the the number of primes $\pi(x)$ less than or equal to x is asymptotic to $x/\ln x$. As the development of this proof proceeds, we intend to improve our library and the interactions with Isabelle’s automated methods, as well as the interactions with Isar’s calculational reasoning. This will ensure that calculations using O notation can be carried out smoothly and naturally. Independently, we still need to formalize the variations on O notation described in Section 2, as well as other asymptotic notions.

A similar treatment of asymptotic notions has been given in Mizar [3, 4] under the “eventually” reading of O notation given in Section 2. This implementation includes Θ and Ω notation as well. But the treatment is less general than the one described here, in that the notions apply only to eventually nonnegative sequences of real numbers.

It seems appropriate to mention here a perennial problem with simple type theory. Our treatment of O notation applies to any type of functions, that is, any collection of functions whose domain and codomain are also types. Since Isabelle’s `typedef` command allows us to turn the real interval $(0, 1)$, say, into a type, we can use O notation directly for functions defined on this interval. *But simple type theory does not allow one to define types that depend on a parameter*, so, for example, one cannot prove theorems involving O notation for functions defined more generally on a real interval (x, y) , where x and y are variables. To do so, one has three options:

1. Work around the problem, for example, using the “on S ” variant of O notation described in Section 2.
2. Replace simple type theory with a formalism that allows dependent types. For example, Coq [11] is based on the Coquand-Huet *calculus of constructions*.
3. Use locales instead of types to fix the subset and parameters that are of interest. (See [2].)

Each option has its drawbacks. The first can be notationally cumbersome; the second involves a more elaborate type theory, with more complex type-theoretic behavior; the third involves reducing one's reliance on the underlying type theory, but giving up the associated notational and computational advantages. We do not yet have a sense what approach, in the long term, is best suited to developing complex mathematical theories.

References

1. Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete mathematics: a foundation for computer science*. Addison-Wesley Publishing Company, Reading, MA, second edition, 1994.
2. F. Kammüller, M. Wenzel, and L. C. Paulson. Locales – a sectioning concept for Isabelle. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Théry, editors, *Proceeding of Theorem Proving in Higher Order Logics, 12th International Conference, TPHOLs'99, Nice, France, September 14 - 17*, volume 1690 of *LNCS*, 1999.
3. Richard Krueger, Piotr Rudnicki, and Paul Shelley. Asymptotic notation. Part I: theory. *Journal of Formalized Mathematics*, Volume 11, 1999. http://mizar.org/JFM/Vol11/asympt_0.html.
4. Richard Krueger, Piotr Rudnicki, and Paul Shelley. Asymptotic notation. Part II: examples and problems. *Journal of Formalized Mathematics*, Volume 11, 1999. http://mizar.org/JFM/Vol11/asympt_1.html.
5. Melvyn B. Nathanson. *Elementary Methods in Number Theory*. Springer, New York, 2000.
6. Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL. A proof assistant for higher-order logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, 2002.
7. Harold N. Shapiro. *Introduction to the theory of numbers*. Pure and Applied Mathematics. John Wiley & Sons Inc., New York, 1983. A Wiley-Interscience Publication.
8. Markus Wenzel. Using axiomatic type classes in Isabelle, 1995. <http://www.cl.cam.ac.uk/Research/HVG/Isabelle/docs.html>.
9. Type classes and overloading in higher-order logic. In E. Gunther and A. Felty, editors, *Proceedings of the 10th international conference on theorem provings in higher-order logic (TPHOLs '97)*, pages 307-322, Murray Hill, New Jersey, 1997.
10. Markus Wenzel and Gertrud Bauer. Calculational reasoning revisited (an Isabelle/Isar experience). In R. J. Boulton and P. B. Jackson, editors, *Theorem Proving in Higher Order Logics: 14th International Conference, TPHOLs 2001, Edinburgh, Scotland, UK, September 3-6, 2001, Proceedings*, volume 2152 of *Lecture Notes in Computer Science*, pages 75–90, 2001.
11. The Coq proof assistant. Developed by the LogiCal project. <http://pauillac.inria.fr/coq/coq-eng.html>.
12. The Isabelle theorem proving environment. Developed by Larry Paulson at Cambridge University and Tobias Nipkow at TU Munich. <http://www.cl.cam.ac.uk/Research/HVG/Isabelle/index.html>.