Formalization of O Notation in Isabelle/HOL

Kevin Donnelly (Jeremy Avigad)
Carnegie Mellon University
July 2004

Asymptotics

First, the motivation:

Theorem: [The Prime Number Theorem]

$$\pi(x) \sim \frac{x}{\ln x}$$

The number of primes less than x is asymptotic to $\frac{x}{\ln x}$.

We are working on formalizing a proof of the prime number theorem using Isabelle/HOL. In support of this project we formalized a very general notion of O notation.

Definition: f is asymptotic to g

$$f(n) \sim g(n) \iff \lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$$

Definition: f is big-o of g

$$f(n) = O\left(g(n)\right) \iff \exists C\ \forall n\ |f(n)| \leq C \cdot |g(n)|$$

This differs slightly from some definitions of O in that it does not rely on having an ordered domain, only an ordered codomain.

Alternative Definitions

Definition: f is big-o of g eventually

$$f(n) = O\left(g(n)\right) \text{ eventually } \iff \underline{\exists m}\ \exists C\ \forall n \underline{\geq m}\ |f(n)| \leq C \cdot |g(n)|$$

Definition: f is big-o of g on S

$$f(n) = O\left(g(n)\right) \text{ on } S \iff \exists m\ \exists C\ \forall n \underline{\in S}\ |f(n)| \leq C \cdot |g(n)|$$

Uses of O notation:

- Computer Science/Algorithms
- Mathematics
  - Number Theory
  - Combinatorics

Examples

- Quicksort sorts in $O(n \log n)$
- $\sum_{i=1}^{n} \frac{1}{i} = \ln n + O(1)$
  (identity used in proving PNT)

O notation in the proof on the PNT:

Definitions:

$$\theta(x) = \sum_{p \le x} \ln p$$

$$\psi(x) = \sum_{p^\alpha \le x} \ln p$$

Lemma:

$$\psi(x) = \theta(x) + O\left(\sqrt{x} \ln x\right)$$

Lemma:

$$\pi(x) = \frac{\theta(x)}{\ln x} + O\left(\frac{x}{\ln^2 x}\right)$$

Theorem:

$$\frac{\pi(x) \ln x}{x} \sim \frac{\theta(x)}{x} \sim \frac{\psi(x)}{x}$$

O notation

Keys to a good formalization of O notation:

- Generality - O notation makes sense on a large range of function types, even on unordered domains.
- Perspicuity - The formalization should support reasoning at a relatively high level.

In addition we must make choices in how to deal with ambiguity and abuse of notation (an $=$ which is not an equivalence!)

Isabelle

Isabelle (developed by Larry Paulson and Tobias Nipkow) is a generic theorem proving framework based on a typed λ-calculus.

Syntax is standard typed lambda calculus, with the addition of sort restrictions on types $(t :: (T :: S))$

Isabelle has several features well-suited to our formalization.

Polymorphism

Isabelle provides powerful polymorphism:

- Parametric polymorphism: $(\alpha)\mathbf{list}, \alpha \Rightarrow \mathbf{bool}$, etc

$$(\lambda f\ g\ x.\ f(g(x))) :: (\rho \Rightarrow \beta) \Rightarrow (\alpha \Rightarrow \rho) \Rightarrow \alpha \Rightarrow \beta$$

- Sort-restricted polymorphism through the use of type classes

$$(\lambda x\ y.\ x \leq y) :: (\alpha :: \mathbf{order}) \Rightarrow (\alpha :: \mathbf{order}) \Rightarrow \mathbf{bool}$$

  where **order** is the class of types on which $\leq$ is defined

Type Classes

Order-sorted type classes, due to Nipkow, provide a more restricted polymorphism.

$$= :: (\alpha :: \mathbf{term}) \Rightarrow (\alpha :: \mathbf{term}) \Rightarrow \mathbf{bool}$$

Type classes form a hierarchy with the pre-defined **logic** class, containing all types, at the top. We can declare types to be a member of a class with arity declarations

$$\begin{array}{rcl} \mathbf{fun} & :: & (\mathbf{logic}, \mathbf{logic})\mathbf{logic} \\ \mathbf{nat}, \mathbf{int}, \mathbf{real} & :: & \mathbf{term} \\ \mathbf{list} & :: & (\mathbf{term})\mathbf{term} \end{array}$$

Type Classes

Type classes can also be used to handle overloading

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

$$
\begin{aligned}
+ &:: (\alpha :: \mathbf{plus}) \Rightarrow \alpha \Rightarrow \alpha \\
1 &:: \alpha :: \mathbf{one}
\end{aligned}
$$

This would declare the constants + and 1 for any type in the class **plus** and **one** respectively.

Axiomatic Type Classes

Type classes can also be given axiomatic restrictions. This is extremely useful in defining general functions like summation over a set.

```
axclass plus_ac0 < plus, zero
commute:"x + y = y + x"
assoc:  "(x + y) + z = x + (y + z)"
zero:  "0 + x = x"
```

$$\text{setsum} :: (\alpha \Rightarrow \beta) \Rightarrow (\alpha)\textbf{set} \Rightarrow (\beta :: \textbf{plus\_ac0})$$

Subclasses of axiomatic classes inherit axioms as expected.

Axiomatic Type Classes

We can use axiomatic type classes to prove generic theorems that will then apply to any type in the class

```
theorem right_zero:  "x + 0 = x::'a::plus_ac0"
```

Since the class was defined axiomatically, we have to prove each type as a member of the class (or each class as a subclass)

```
instance semiring < plus_ac0
```

```
instance nat ::  semiring
```

Isabelle/HOL

Isabelle/HOL is a formalization of higher order logic similar to the Church's Simple Theory of Types with polymorphism It provides

- higher order equality: $=$
- the familiar logical operations and quantifiers: $\forall, \exists, \rightarrow, \&, |, \tilde{}, \exists!$
- base types: **nat**, **bool**, **int**
- constructed types: $\alpha \times \beta, (\alpha)\mathbf{set}, (\alpha, \beta)\mathbf{fun}$
- a set theory similar to Russel and Whitehead's Theory of Classes
- nice automated theorem proving and simplification tactics
- The **ring** and **ordered_ring** axiomatic type classes (Bauer, Wenzel and Paulson)

HOL-Complex

HOL-Complex is a formalization of parts of analysis, due to Jacques Fleuriot, in Isabelle/HOL which provides

- The type **real** of real numbers and associated operations and functions: $+, -, *, {}^{-1}, \log, \ln, e\hat{}$, etc
- Derivatives and Integrals
- A summation operator over **nat** $\Rightarrow$ **real** function types well suited to things like infinite sums

$$\text{sumr} :: \mathbf{nat} \Rightarrow \mathbf{nat} \Rightarrow (\mathbf{nat} \Rightarrow \mathbf{real}) \Rightarrow \mathbf{real}$$

$$\text{sumr}\ n\ m\ f = \sum_{n \leq x < m} f(x)$$

Formalizing O notation

O formulas are not really equations.

$$\begin{aligned} f(x) &= x \\ f(x) &= O(x) \\ f(x) &= O\left(x^2\right) \\ O\left(x^2\right) &\neq O(x) \end{aligned}$$

Ambiguity

O notation is ambiguous.

While it presents itself as a function on terms, it is really a higher order function, on a lambda term with an implicit binder:

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

is true if we read it as

$$\lambda x.\ ax^2 + bx + c = O\left(\lambda x.\ x^2\right)$$

but not as

$$\lambda b.\ ax^2 + bx + c = O\left(\lambda b.\ x^2\right)$$

Solution: set inclusion and higher order function

$f(x) = O\left(g(x)\right)$ really means $f \in O\left(g\right)$ where $O\left(g\right)$ is the set of all functions bounded by a constant multiple of g.

I will use this notation from now on

Definition:

$$O\left(g\right) = \{h \mid \exists C\ \forall x\ |h(x)| \le C * |g(x)|\}$$

In order to make it as general as possible, we define O on functions from any type into a (non-degenerate) ordered ring.

$$O :: (\alpha \Rightarrow \beta :: \textbf{ordered\_ring}) \Rightarrow (\alpha \Rightarrow \beta)\textbf{set}$$

This is enough machinery to prove a few simple things like $f \in O(f)$ but to formalize something more complex like

$$“\sum_{i=1}^{n} \frac{1}{i} = \ln n + O(1)”$$

and to make our O notation usable easily in proofs, we need more. Specifically, we need arithmetic operations functions, set and elements

Defining Arithmetic Operations

We want to define *, +, etc on functions of type $\alpha \Rightarrow \beta$ and sets of type (β)**set** such that these operations are defined on β

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

This is simply asserts the existence a function of the right type with the corresponding symbol $(*)$.

We then give that symbol a definition

```
defs
func_times:  "f * g == (λ x. (f x) * (g x))"
set_times:  "A * B == {c | ∃a∈A.∃b∈B.c = a * b}"
```

Similarly we declare **fun** and **set** in the classes **plus** and **minus** and provide similar definitions for the constants + and −

We then define a zero for both classes

```
instance fun :: (type,zero)zero
instance set :: (zero)zero
defs
func_zero: "0::('a => 'b::zero) == (λx. 0::'b)"
set_zero: "0::('a::zero)set == {0::'a}"
```

And now we can prove each of these classes in **plus_ac0**

```
instance fun :: (type,plus_ac0)plus_ac0
instance set :: (plus_ac0)plus_ac0
```

Also, in order to facilitate easier use of O notation we define the arithmetic functions that take an element and set argument

```
constdefs
elt_set_plus::"'a::plus => 'a set => 'a set" (infixl
"+o" 70)
"a +o B == {c | ∃b∈B. c = a + b}"
```

$$+o :: (\alpha \Rightarrow \beta) \Rightarrow (\alpha \Rightarrow \beta)\mathbf{set} \Rightarrow (\alpha \Rightarrow \beta)\mathbf{set}$$

We similarly define $*o$ and $-o$

O Formulas

We now have enough to formally state a wide range of O “equations"

The standard form is

$$f \in g +\mathrm{o}\ O\,(h)$$

This form suffices to express almost any statement of O notation (and all that we need for the PNT) so most of the theorems we have proved about O formulas are proved about formulas of this form.

$$\text{“}\sum_{i=1}^{n} \frac{1}{i} = \ln n + O\,(1)\text{”}$$

Can be stated in this form as

$$\left(\lambda n. \sum_{i=1}^{n} \frac{1}{i}\right) \in \ln\ +\mathrm{o}\ O\,(\lambda n.\ 1)$$

In Isabelle syntax

```
theorem sum_inverse_eq_ln_1:
"(λn.sumr 0 n (λx.1/(x + 1))) ∈ (λn.ln (real (n + 1)))
+o O(λn.1)"
```

$$\left(\lambda n.\sum_{i=1}^{n}\frac{1}{i}\right) \in \ln\ +_{\mathrm{o}}\ O\left(\lambda n.\ 1\right)$$

This is slightly more cumbersome than standard O notation because you have to convert terms in the equation into functions, but this is really always part of O notation, it is just left implicit.

O Variations

$$O\,(g) = \{h \mid \exists C \; \forall x \; |h(x)| \leq C * |g(x)|\}$$

Interpreting the O as a function from functions to function sets also lets us easily handle other interpretations of O notation. One such other interpretation would be, on an ordered domain:

$$O\,(g) \text{ eventually} = \{h \mid \exists C \; \exists n \; \forall x > n \; |h(x)| \leq C * |g(x)|\}$$

Another would restrict the set of interest as a subset of the domain, as in:

$$O\,(g) \text{ on } S = \{h \mid \exists C \; \forall x \in S \; |h(x)| \leq C * |g(x)|\}$$

We can get both of these variations just by adding a function from function sets to function sets!

We introduce the weakly binding postfix function

$$\text{eventually} :: ((\alpha :: \textbf{linorder}) \Rightarrow \beta)\textbf{set} \Rightarrow (\alpha \Rightarrow \beta)\textbf{set}$$

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

Which we can use to get fairly textbook looking O formulas

$$\lambda x.\ x^2 \in O\,(\lambda x.x + 1)\,\text{eventually}$$

We also introduce the binary

$$\text{on} :: (\alpha \Rightarrow \beta)\textbf{set} \Rightarrow (\alpha)\textbf{set} \Rightarrow (\alpha \Rightarrow \beta)\textbf{set}$$

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

Using O notation

In order to use our O notation in proofs there are two important classes of lemmas that we proved.

- manipulating sets and elements
- asymptotic properties

Manipulating set and elements

Normalization

set-plus-rearrange	$(a + C) + (b + D) = (a + b) + (C + D)$
set-plus-rearrange2	$a + (b + C) = (a + b) + C$
set-plus-rearrange3	$(a + C) + D = a + (C + D)$
set-plus-rearrange4	$C + (a + D) = a + (C + D)$

These rewrite rules give us a term of the form

$(a + b + ...)$ +o $(O\,(a') + O\,(b') + ...)$

Example:

```
theorem set-rearrange:
"(f +o O(h)) + (g +o O(i)) = (f + g) +o (O(h) + O(i))"
by(simp only:  set-plus-rearranges plus-ac0)
```

Monotonicity of arithmetic operations over sets and elements

set-plus-intro	$[\|a \in C, b \in D\|] \Rightarrow a + b \in C + D$
set-plus-intro2	$b \in C \Rightarrow a + b \in a + C$
set-zero-plus	$0 + C = C$
set-plus-mono	$C \subseteq D \Rightarrow a + C \subseteq a + D$
set-plus-mono2	$[\|C \subseteq D, E \subseteq F\|] \Rightarrow C + E \subseteq D + F$
set-plus-mono3	$a \in C \Rightarrow a + D \subseteq C + D$
set-plus-mono4	$a \in C \Rightarrow a + D \subseteq D + C$

Asymptotic properties

Direct set-theoretic properties of O sets

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

Addition properties of O sets

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

```
theorem bigo_bounded2:  "[|∀n.(lb n <= x n) & (x n <= lb
n + f n); f∈O(g)|] ==> x∈(lb + f) +o O(g)"
```

This last theorem lets us prove that a function is in an O set by proving appropriate lower and upper bounds for the function. This is the method used to prove

$$\left(\lambda n. \sum_{i=1}^{n} \frac{1}{i}\right) \in \ln \; +\mathrm{o} \; O\,(\lambda n.\; 1)$$

References

- Knuth et. al. *Concrete Mathematics*, 2nd Edition, Ch. 9.
- Paulson, Lawrence C. "Introduction to Isabelle," http://www.cl.cam.ac.uk/Research/HVG/Isabelle/dist/Isabelle2004/doc/intro.pdf
- Nipkow, Tobias and Lawrence C. Paulson and Markus Wenzel. "Isabelle's Logics: HOL," http://www.cl.cam.ac.uk/Research/HVG/Isabelle/dist/Isabelle2004/doc/logics-HOL.pdf
- Wenzel, Markus. "Using axiomatic type classes in Isabelle," http://www.cl.cam.ac.uk/Research/HVG/Isabelle/dist/Isabelle2004/doc/axclass.pdf