

Implementing the Goodstein Function in λ -Calculus

Bertram Felgenhauer

October 13, 2025

Abstract

In this formalization, we develop an implementation of the Goodstein function \mathcal{G} in plain λ -calculus, linked to a concise, self-contained specification. The implementation works on a Church-encoded representation of countable ordinals. The initial conversion to hereditary base 2 is not covered, but the material is sufficient to compute the particular value $\mathcal{G}(16)$, and easily extends to other fixed arguments.

Contents

1	Introduction	2
2	Specification	3
2.1	Hereditary base representation	3
2.2	The Goodstein function	4
3	Ordinals	4
3.1	Evaluation	5
3.2	Goodstein function and sequence	5
3.3	Properties of evaluation	6
3.4	Arithmetic properties	6
4	Cantor normal form	7
4.1	Conversion to and from the ordinal type <i>Ord</i>	7
4.2	Evaluation	8
4.3	Transfer of the <i>Ord</i> induction principle to <i>C</i>	8
4.4	Goodstein function and sequence on <i>C</i>	8
4.5	Properties	9
5	Hereditary base <i>b</i> representation	10
5.1	Uniqueness	10
5.2	Correctness of <i>stepC</i>	11
5.3	Surjectivity of <i>evalC</i>	11
5.4	Monotonicity of <i>hbase</i>	12
5.5	Conversion to and from <i>nat</i>	12

6	The Goodstein function revisited	13
7	Translation to λ-calculus	13
7.1	Alternative: free theorems	14

1 Introduction

Given a number n and a base b , we can write n in *hereditary base b* , which results from writing n in base b , and then each exponent in hereditary base b again. For example, 7 in hereditary base 3 is $3^1 \cdot 2 + 1$. Given the hereditary base b representation of n , we can reinterpret it in base $b + 1$ by replacing all occurrences of b by $b + 1$.

The Goodstein sequence starting at n in base 2 is obtained by iteratively taking a number in hereditary base b , reinterpreting it in base $b + 1$, and subtracting 1. The next step is the same with b incremented by 1, and so on. So starting for example at 4, we compute

$$\begin{aligned}
4 &= 2^{2^1} \rightarrow 3^{3^1} - 1 = 26 \\
26 &= 3^2 \cdot 2 + 3^1 \cdot 2 + 2 \rightarrow 4^2 \cdot 2 + 4^1 \cdot 2 + 1 \cdot 2 - 1 = 41 \\
41 &= 4^2 \cdot 2 + 4^1 \cdot 2 + 1 \rightarrow 5^2 \cdot 2 + 5^1 \cdot 2 + 1 - 1 = 60
\end{aligned}$$

and so on. We stop when we reach 0. Goodstein's theorem states that this process always terminates [3]. This result is independent of Peano Arithmetic, and is intimately connected to countable ordinals and the slow growing hierarchy (e.g., the Hardy function) [2]. The length of the resulting sequence is the Goodstein function, denoted by $\mathcal{G}(n)$. For example, $\mathcal{G}(3) = 6$.

For this formalization, we are interested in implementing the Goodstein function in λ -calculus. More concretely, we want to define the value $\mathcal{G}(16)$ (which is huge; for example, it exceeds Graham's number), in order to bound its Kolmogorov complexity. Our concrete measure of Kolmogorov complexity is the program length in the Binary Lambda Calculus [4, 5]. It turns out that we can define $\mathcal{G}(16)$ as follows, giving a complexity bound of 195 bits.

$$\begin{aligned}
exp\omega &= (\lambda z \ s \ l. \ n \ s \ (\lambda x \ z. \ l \ (\lambda n. \ n \ x \ z)) \ (\lambda f \ z. \ l \ (\lambda n. \ f \ n \ z)) \ z) \\
goodstein &= (\lambda n \ c. \ n \\
&\quad (\lambda x. \ x) \\
&\quad (\lambda n \ m. \ n \ (\lambda f \ x. \ m \ f \ (f \ x))) \\
&\quad (\lambda f \ m. \ f \ (\lambda f \ x. \ m \ f \ (f \ (f \ x))) \ m) \\
&\quad c) \\
\mathcal{G}_{16} &= (\lambda e. \ goodstein \ (e \ (e \ (e \ (e \ (\lambda z \ s \ l. \ z)))) \ (\lambda x. \ x)) \ exp\omega
\end{aligned}$$

We rely on a shallow embedding of the λ -calculus throughout the formalization, so it turns out that we cannot quite prove this claim in Isabelle/HOL;

the expression for \mathcal{G}_{16} cannot be typed. However, we can prove that the building blocks *exp ω* and *goodstein* work correctly in the sense that

- *exp ω^4* ($\lambda z \text{ s l. } z$) is the hereditary base 2 representation of 16; and
- *goodstein c n* computes the length of a Goodstein sequence given that the hereditary base $c+1$ representation of the c -th value in the sequence is equal to n .

The remaining steps are easily verified by a human.

Contributions. Our main contributions are a concise specification of the Goodstein function, another proof of Goodstein’s theorem, and establishing the connection to λ -calculus as already outlined.

Related work. There is already a formalization of Goodstein’s theorem in the AFP entry on nested multisets [1], which comes with a formalization of ordinal arithmetic. Our focus is different, since our goal is to obtain an implementation of the Goodstein function in λ -calculus. Most notably, the intermediate type *Ord* that we use to represent ordinal numbers has far more structure than the ordinals themselves. In particular it can represent arbitrary trees; if we were to compute $\omega + 1$, $1 + \omega$ and ω on this type, we would get three different results. However, we will use the operations such that $1 + \omega$ is never computed, keeping the connection to countable ordinals intact. Proving this is a large, albeit hidden, part of our formalization.

Acknowledgement. John Tromp raised the question of a concise λ -calculus term computing $\mathcal{G}(16)$. He also provided feedback on a draft version of this document.

2 Specification

```
theory Goodstein-Lambda
  imports Main
begin
```

2.1 Hereditary base representation

We define a data type of trees and an evaluation function that sums siblings and exponentiates with respect to the given base on nesting.

```
datatype C = C (unC: C list)
```

```
fun evalC where
  evalC b (C []) = 0
| evalC b (C (x # xs)) = b ^ evalC b x + evalC b (C xs)
```

value $evalC\ 2\ (C\ []) = 0$
value $evalC\ 2\ (C\ [C\ []]) = 2^0 = 1$
value $evalC\ 2\ (C\ [C\ [C\ []]]) = 2^1 = 2$
value $evalC\ 2\ (C\ [C\ [], C\ []]) = 2^0 + 2^0 = 2^0 \cdot 2 = 2$; not in hereditary base 2

The hereditary base representation is characterized as trees (i.e., nested lists) whose lists have monotonically increasing evaluations, with fewer than b repetitions for each value. We will show later that this representation is unique.

inductive-set $hbase$ for b where

$C\ [] \in hbase\ b$
 $| i \neq 0 \implies i < b \implies n \in hbase\ b \implies$
 $C\ ms \in hbase\ b \implies (\bigwedge m'. m' \in set\ ms \implies evalC\ b\ n < evalC\ b\ m') \implies$
 $C\ (replicate\ i\ n\ @\ ms) \in hbase\ b$

We can convert to and from natural numbers as follows.

definition $H2N$ where

$H2N\ b\ n = evalC\ b\ n$

As we will show later, $H2N\ b$ restricted to $hbase\ n$ is bijective if $2 \leq b$, so we can convert from natural numbers by taking the inverse.

definition $N2H$ where

$N2H\ b\ n = inv-into\ (hbase\ b)\ (H2N\ b)\ n$

2.2 The Goodstein function

We define a function that computes the length of the Goodstein sequence whose c -th element is $g_c = n$. Termination will be shown later, thereby establishing Goodstein's theorem.

function (*sequential*) $goodstein :: nat \Rightarrow nat \Rightarrow nat$ where

$goodstein\ 0\ n = 0$
 — we start counting at 1; also note that the initial base is $c + 1$ and
 — hereditary base 1 makes no sense, so we have to avoid this case
 $| goodstein\ c\ 0 = c$
 $| goodstein\ c\ n = goodstein\ (c+1)\ (H2N\ (c+2)\ (N2H\ (c+1)\ n) - 1)$
 $\langle proof \rangle$

abbreviation \mathcal{G} where

$\mathcal{G}\ n \equiv goodstein\ (Suc\ 0)\ n$

3 Ordinals

The following type contains countable ordinals, by the usual case distinction into 0, successor ordinal, or limit ordinal; limit ordinals are given by their

fundamental sequence. Hereditary base b representations carry over to such ordinals by replacing each occurrence of the base by ω .

datatype $Ord = Z \mid S\ Ord \mid L\ nat \Rightarrow Ord$

Note that the following arithmetic operations are not correct for all ordinals. However, they will only be used in cases where they actually correspond to the ordinal arithmetic operations.

primrec $addO$ **where**

$addO\ n\ Z = n$
 $| addO\ n\ (S\ m) = S\ (addO\ n\ m)$
 $| addO\ n\ (L\ f) = L\ (\lambda i. addO\ n\ (f\ i))$

primrec $mulO$ **where**

$mulO\ n\ Z = Z$
 $| mulO\ n\ (S\ m) = addO\ (mulO\ n\ m)\ n$
 $| mulO\ n\ (L\ f) = L\ (\lambda i. mulO\ n\ (f\ i))$

definition ω **where**

$\omega = L\ (\lambda n. (S\ \frown n)\ Z)$

primrec $exp\omega$ **where**

$exp\omega\ Z = S\ Z$
 $| exp\omega\ (S\ n) = mulO\ (exp\omega\ n)\ \omega$
 $| exp\omega\ (L\ f) = L\ (\lambda i. exp\omega\ (f\ i))$

3.1 Evaluation

Evaluating an ordinal number at base b is accomplished by taking the b -th element of all fundamental sequences and interpreting zero and successor over the natural numbers.

primrec $evalO$ **where**

$evalO\ b\ Z = 0$
 $| evalO\ b\ (S\ n) = Suc\ (evalO\ b\ n)$
 $| evalO\ b\ (L\ f) = evalO\ b\ (f\ b)$

3.2 Goodstein function and sequence

We can define the Goodstein function very easily, but proving correctness will take a while.

primrec $goodsteinO$ **where**

$goodsteinO\ c\ Z = c$
 $| goodsteinO\ c\ (S\ n) = goodsteinO\ (c+1)\ n$
 $| goodsteinO\ c\ (L\ f) = goodsteinO\ c\ (f\ (c+2))$

primrec $stepO$ **where**

$stepO\ c\ Z = Z$
 $| stepO\ c\ (S\ n) = n$

| $stepO\ c\ (L\ f) = stepO\ c\ (f\ (c+2))$

We can compute a few values of the Goodstein sequence starting at 4.

definition $g4O$ **where**

$g4O\ n = fold\ stepO\ [1..<Suc\ n]\ ((exp\omega\ \frown\ 3)\ Z)$

value $map\ (\lambda n. evalO\ (n+2)\ (g4O\ n))\ [0..<10]$
 $\text{---}\ [4,\ 26,\ 41,\ 60,\ 83,\ 109,\ 139,\ 173,\ 211,\ 253]$

3.3 Properties of evaluation

lemma $evalO\text{-}addO\ [simp]:$

$evalO\ b\ (addO\ n\ m) = evalO\ b\ n + evalO\ b\ m$
 $\langle proof \rangle$

lemma $evalO\text{-}mulO\ [simp]:$

$evalO\ b\ (mulO\ n\ m) = evalO\ b\ n * evalO\ b\ m$
 $\langle proof \rangle$

lemma $evalO\text{-}n\ [simp]:$

$evalO\ b\ ((S\ \frown\ n)\ Z) = n$
 $\langle proof \rangle$

lemma $evalO\text{-}\omega\ [simp]:$

$evalO\ b\ \omega = b$
 $\langle proof \rangle$

lemma $evalO\text{-}exp\omega\ [simp]:$

$evalO\ b\ (exp\omega\ n) = b^{\frown (evalO\ b\ n)}$
 $\langle proof \rangle$

Note that evaluation is useful for proving that *Ord* values are distinct:

notepad begin

$\langle proof \rangle$

end

3.4 Arithmetic properties

lemma $addO\text{-}Z\ [simp]:$

$addO\ Z\ n = n$
 $\langle proof \rangle$

lemma $addO\text{-}assoc\ [simp]:$

$addO\ n\ (addO\ m\ p) = addO\ (addO\ n\ m)\ p$
 $\langle proof \rangle$

lemma $mulO\text{-}distrib\ [simp]:$

$mulO\ n\ (addO\ p\ q) = addO\ (mulO\ n\ p)\ (mulO\ n\ q)$
 $\langle proof \rangle$

lemma *mulO-assoc* [simp]:

$$\text{mulO } n \ (\text{mulO } m \ p) = \text{mulO } (\text{mulO } n \ m) \ p$$
 $\langle \text{proof} \rangle$

lemma *expw-addO* [simp]:

$$\text{expw } (\text{addO } n \ m) = \text{mulO } (\text{expw } n) \ (\text{expw } m)$$
 $\langle \text{proof} \rangle$

4 Cantor normal form

The previously introduced tree type C can be used to represent Cantor normal forms; they are trees (evaluated at base ω) such that siblings are in non-decreasing order. One can think of this as hereditary base ω . The plan is to mirror selected operations on ordinals in Cantor normal forms.

4.1 Conversion to and from the ordinal type Ord

fun *C2O* **where**

$$C2O \ (C \ []) = Z$$

$$| \ C2O \ (C \ (n \ \# \ ns)) = \text{addO } (C2O \ (C \ ns)) \ (\text{expw } (C2O \ n))$$

definition *O2C* **where**

$$O2C = \text{inv } C2O$$

We show that $C2O$ is injective, meaning the inverse is unique.

lemma *addO-expw-inj*:
assumes $\text{addO } n \ (\text{expw } m) = \text{addO } n' \ (\text{expw } m')$
shows $n = n'$ **and** $m = m'$
 $\langle \text{proof} \rangle$

lemma *C2O-inj*:

$$C2O \ n = C2O \ m \implies n = m$$
 $\langle \text{proof} \rangle$

lemma *O2C-C2O* [simp]:

$$O2C \ (C2O \ n) = n$$
 $\langle \text{proof} \rangle$

lemma *O2C-Z* [simp]:

$$O2C \ Z = C \ []$$
 $\langle \text{proof} \rangle$

lemma *C2O-replicate*:

$$C2O \ (C \ (\text{replicate } i \ n)) = \text{mulO } (\text{expw } (C2O \ n)) \ ((S \ \frown \ i) \ Z)$$
 $\langle \text{proof} \rangle$

lemma *C2O-app*:

$C2O (C (xs @ ys)) = addO (C2O (C ys)) (C2O (C xs))$
 $\langle proof \rangle$

4.2 Evaluation

lemma *evalC-def'*:
 $evalC b n = evalO b (C2O n)$
 $\langle proof \rangle$

lemma *evalC-app [simp]*:
 $evalC b (C (ns @ ms)) = evalC b (C ns) + evalC b (C ms)$
 $\langle proof \rangle$

lemma *evalC-replicate [simp]*:
 $evalC b (C (replicate c n)) = c * evalC b (C [n])$
 $\langle proof \rangle$

4.3 Transfer of the *Ord* induction principle to C

fun *funC* **where** — *funC* computes the fundamental sequence on C
 $funC (C []) = (\lambda i. [C []])$
 $| funC (C (C [] \# ns)) = (\lambda i. replicate i (C ns))$
 $| funC (C (n \# ns)) = (\lambda i. [C (funC n i @ ns)])$

lemma *C2O-cons*:
 $C2O (C (n \# ns)) =$
 $(if n = C [] then S (C2O (C ns)) else L (\lambda i. C2O (C (funC n i @ ns))))$
 $\langle proof \rangle$

lemma *C-Ord-induct*:
assumes $P (C [])$
and $\bigwedge ns. P (C ns) \implies P (C (C [] \# ns))$
and $\bigwedge n ns ms. (\bigwedge i. P (C (funC (C (n \# ns)) i @ ms))) \implies$
 $P (C (C (n \# ns) \# ms))$
shows $P n$
 $\langle proof \rangle$

4.4 Goodstein function and sequence on C

function (*domintros*) *goodsteinC* **where**
 $goodsteinC c (C []) = c$
 $| goodsteinC c (C (C [] \# ns)) = goodsteinC (c+1) (C ns)$
 $| goodsteinC c (C (C (n \# ns) \# ms)) =$
 $goodsteinC c (C (funC (C (n \# ns)) (c+2) @ ms))$
 $\langle proof \rangle$

termination
 $\langle proof \rangle$

lemma *goodsteinC-def'*:

$goodsteinC\ c\ n = goodsteinO\ c\ (C2O\ n)$
 $\langle proof \rangle$

function (*domintros*) *stepC* **where**
 $stepC\ c\ (C\ []) = C\ []$
 $| stepC\ c\ (C\ (C\ []\ \# ns)) = C\ ns$
 $| stepC\ c\ (C\ (C\ (n\ \# ns)\ \# ms)) =$
 $stepC\ c\ (C\ (funC\ (C\ (n\ \# ns))\ (Suc\ (Suc\ c))\ @\ ms))$
 $\langle proof \rangle$

termination
 $\langle proof \rangle$

definition g_4C **where**
 $g_4C\ n = fold\ stepC\ [1..<Suc\ n]\ (C\ [C\ [C\ [C\ []]])$

value $map\ (\lambda n. evalC\ (n+2)\ (g_4C\ n))\ [0..<10]$
 $— [4, 26, 41, 60, 83, 109, 139, 173, 211, 253]$

4.5 Properties

lemma *stepC-def'*:
 $stepC\ c\ n = O2C\ (stepO\ c\ (C2O\ n))$
 $\langle proof \rangle$

lemma *funC-ne* [*simp*]:
 $funC\ m\ (Suc\ n) \neq []$
 $\langle proof \rangle$

lemma *evalC-funC* [*simp*]:
 $evalC\ b\ (C\ (funC\ n\ b)) = evalC\ b\ (C\ [n])$
 $\langle proof \rangle$

lemma *stepC-app* [*simp*]:
 $n \neq C\ [] \implies stepC\ c\ (C\ (unC\ n\ @\ ns)) = C\ (unC\ (stepC\ c\ n)\ @\ ns)$
 $\langle proof \rangle$

lemma *stepC-cons* [*simp*]:
 $ns \neq [] \implies stepC\ c\ (C\ (n\ \# ns)) = C\ (unC\ (stepC\ c\ (C\ [n]))\ @\ ns)$
 $\langle proof \rangle$

lemma *stepC-dec*:
 $n \neq C\ [] \implies Suc\ (evalC\ (Suc\ (Suc\ c))\ (stepC\ c\ n)) = evalC\ (Suc\ (Suc\ c))\ n$
 $\langle proof \rangle$

lemma *stepC-dec'*:
 $n \neq C\ [] \implies evalC\ (c+3)\ (stepC\ c\ n) < evalC\ (c+3)\ n$
 $\langle proof \rangle$

5 Hereditary base b representation

We now turn to properties of the *hbase* b subset of trees.

5.1 Uniqueness

We show uniqueness of the hereditary base representation by showing that *evalC* b restricted to *hbase* b is injective.

lemma *hbaseI2*:

$i < b \implies n \in \text{hbase } b \implies C \ m \in \text{hbase } b \implies$
 $(\bigwedge m'. m' \in \text{set } m \implies \text{evalC } b \ n < \text{evalC } b \ m') \implies$
 $C \ (\text{replicate } i \ n \ @ \ m) \in \text{hbase } b$
 $\langle \text{proof} \rangle$

lemmas *hbase-singletonI* =

hbase.intros(2)[of 1 Suc (Suc b) for b, OF - - hbase.intros(1), simplified]

lemma *hbase-hd*:

$C \ ns \in \text{hbase } b \implies ns \neq [] \implies \text{hd } ns \in \text{hbase } b$
 $\langle \text{proof} \rangle$

lemmas *hbase-hd' [dest] = hbase-hd[of n # ns for n ns, simplified]*

lemma *hbase-tl*:

$C \ ns \in \text{hbase } b \implies ns \neq [] \implies C \ (\text{tl } ns) \in \text{hbase } b$
 $\langle \text{proof} \rangle$

lemmas *hbase-tl' [dest] = hbase-tl[of n # ns for n ns, simplified]*

lemma *hbase-elt [dest]*:

$C \ ns \in \text{hbase } b \implies n \in \text{set } ns \implies n \in \text{hbase } b$
 $\langle \text{proof} \rangle$

lemma *evalC-sum-list*:

$\text{evalC } b \ (C \ ns) = \text{sum-list } (\text{map } (\lambda n. b^{\wedge} \text{evalC } b \ n) \ ns)$
 $\langle \text{proof} \rangle$

lemma *sum-list-replicate*:

$\text{sum-list } (\text{replicate } n \ x) = n * x$
 $\langle \text{proof} \rangle$

lemma *base-red*:

fixes $b :: \text{nat}$
assumes $n: \bigwedge n'. n' \in \text{set } ns \implies n < n' \ i < b \ i \neq 0$
and $m: \bigwedge m'. m' \in \text{set } ms \implies m < m' \ j < b \ j \neq 0$
and $s: i * b^{\wedge} n + \text{sum-list } (\text{map } (\lambda n. b^{\wedge} n) \ ns) = j * b^{\wedge} m + \text{sum-list } (\text{map } (\lambda n. b^{\wedge} n) \ ms)$
shows $i = j \wedge n = m$

$\langle \text{proof} \rangle$

lemma *evalC-inj-on-hbase*:

$n \in \text{hbase } b \implies m \in \text{hbase } b \implies \text{evalC } b \ n = \text{evalC } b \ m \implies n = m$

$\langle \text{proof} \rangle$

5.2 Correctness of *stepC*

We show that *stepC* *c* preserves hereditary base $c + 2$ representations. In order to cover intermediate results produced by *stepC*, we extend the hereditary base representation to allow the least significant digit to be equal to *b*, which essentially means that we may have an extra sibling in front on every level.

inductive-set *hbase-ext* **for** *b* **where**

$n \in \text{hbase } b \implies n \in \text{hbase-ext } b$

| $n \in \text{hbase-ext } b \implies$

$C \ m \in \text{hbase } b \implies (\bigwedge m'. m' \in \text{set } m \implies \text{evalC } b \ n \leq \text{evalC } b \ m') \implies$

$C \ (n \# m) \in \text{hbase-ext } b$

lemma *hbase-ext-hd'* [*dest*]:

$C \ (n \# ns) \in \text{hbase-ext } b \implies n \in \text{hbase-ext } b$

$\langle \text{proof} \rangle$

lemma *hbase-ext-tl*:

$C \ ns \in \text{hbase-ext } b \implies ns \neq [] \implies C \ (tl \ ns) \in \text{hbase } b$

$\langle \text{proof} \rangle$

lemmas *hbase-ext-tl'* [*dest*] = *hbase-ext-tl*[*of* *n* # *ns* **for** *n* *ns*, *simplified*]

lemma *hbase-funC*:

$c \neq 0 \implies C \ (n \# ns) \in \text{hbase-ext } (Suc \ c) \implies$

$C \ (\text{funC } n \ (Suc \ c) \ @ \ ns) \in \text{hbase-ext } (Suc \ c)$

$\langle \text{proof} \rangle$

lemma *stepC-sound*:

$n \in \text{hbase-ext } (Suc \ (Suc \ c)) \implies \text{stepC } c \ n \in \text{hbase } (Suc \ (Suc \ c))$

$\langle \text{proof} \rangle$

5.3 Surjectivity of *evalC*

Note that the base must be at least 2.

lemma *evalC-surjective*:

$\exists n' \in \text{hbase } (Suc \ (Suc \ b)). \text{evalC } (Suc \ (Suc \ b)) \ n' = n$

$\langle \text{proof} \rangle$

5.4 Monotonicity of $hbase$

Here we show that every hereditary base b number is also a valid hereditary base $b + 1$ number. This is not immediate because we have to show that monotonicity of siblings is preserved.

lemma *hbase-evalC-mono*:

assumes $n \in hbase\ b\ m \in hbase\ b\ evalC\ b\ n < evalC\ b\ m$

shows $evalC\ (Suc\ b)\ n < evalC\ (Suc\ b)\ m$

<proof>

lemma *hbase-mono*:

$n \in hbase\ b \implies n \in hbase\ (Suc\ b)$

<proof>

5.5 Conversion to and from nat

We have previously defined $H2N\ b = evalC\ b$ and $N2H\ b$ as its inverse. So we can use the injectivity and surjectivity of $evalC\ b$ for simplification.

lemma *N2H-inv*:

$n \in hbase\ b \implies N2H\ b\ (H2N\ b\ n) = n$

<proof>

lemma *H2N-inv*:

$H2N\ (Suc\ (Suc\ b))\ (N2H\ (Suc\ (Suc\ b))\ n) = n$

<proof>

lemma *N2H-eqI*:

$n \in hbase\ (Suc\ (Suc\ b)) \implies$

$H2N\ (Suc\ (Suc\ b))\ n = m \implies N2H\ (Suc\ (Suc\ b))\ m = n$

<proof>

lemma *N2H-neI*:

$n \in hbase\ (Suc\ (Suc\ b)) \implies$

$H2N\ (Suc\ (Suc\ b))\ n \neq m \implies N2H\ (Suc\ (Suc\ b))\ m \neq n$

<proof>

lemma *N2H-0 [simp]*:

$N2H\ (Suc\ (Suc\ c))\ 0 = C\ []$

<proof>

lemma *N2H-nz [simp]*:

$0 < n \implies N2H\ (Suc\ (Suc\ c))\ n \neq C\ []$

<proof>

6 The Goodstein function revisited

We are now ready to prove termination of the Goodstein function *goodstein* as well as its relation to *goodsteinC* and *goodsteinO*.

lemma *goodstein-aux*:

$$\begin{aligned} & \text{goodsteinC } (Suc\ c) (N2H\ (Suc\ (Suc\ c))\ (Suc\ n)) = \\ & \text{goodsteinC } (c+2) (N2H\ (c+3) (H2N\ (c+3) (N2H\ (c+2) (n+1)) - 1)) \\ & \langle proof \rangle \end{aligned}$$

termination *goodstein*

$\langle proof \rangle$

lemma *goodstein-def'*:

$$\begin{aligned} & c \neq 0 \implies \text{goodstein } c\ n = \text{goodsteinC } c\ (N2H\ (c+1)\ n) \\ & \langle proof \rangle \end{aligned}$$

lemma *goodstein-impl*:

$$\begin{aligned} & c \neq 0 \implies \text{goodstein } c\ n = \text{goodsteinO } c\ (C2O\ (N2H\ (c+1)\ n)) \\ & \text{— but note that } N2H \text{ is not executable as currently defined} \\ & \langle proof \rangle \end{aligned}$$

lemma *goodstein-16*:

$$\begin{aligned} & G\ 16 = \text{goodsteinO } 1\ (\text{expw } (\text{expw } (\text{expw } (\text{expw } Z)))) \\ & \langle proof \rangle \end{aligned}$$

7 Translation to λ -calculus

We define Church encodings for *nat* and *Ord*. Note that we are basically in a Hindley-Milner type system, so we cannot use a proper polymorphic type. We can still express Church encodings as folds over values of the original type.

abbreviation Z_N **where** $Z_N \equiv (\lambda s\ z. z)$

abbreviation S_N **where** $S_N \equiv (\lambda n\ s\ z. s\ (n\ s\ z))$

primrec *fold-nat* ($\langle \cdot \rangle_N$) **where**

$$\begin{aligned} & \langle 0 \rangle_N = Z_N \\ & | \langle Suc\ n \rangle_N = S_N\ \langle n \rangle_N \end{aligned}$$

lemma *one_N*:

$$\begin{aligned} & \langle 1 \rangle_N = (\lambda x. x) \\ & \langle proof \rangle \end{aligned}$$

abbreviation Z_O **where** $Z_O \equiv (\lambda z\ s\ l. z)$

abbreviation S_O **where** $S_O \equiv (\lambda n\ z\ s\ l. s\ (n\ z\ s\ l))$

abbreviation L_O **where** $L_O \equiv (\lambda f\ z\ s\ l. l\ (\lambda i. f\ i\ z\ s\ l))$

primrec *fold-Ord* ($\langle \cdot \rangle_O$) **where**

$$\begin{aligned}
& \langle Z \rangle_O = Z_O \\
& | \langle S \ n \rangle_O = S_O \ \langle n \rangle_O \\
& | \langle L \ f \rangle_O = L_O \ (\lambda i. \langle f \ i \rangle_O)
\end{aligned}$$

The following abbreviations and lemmas show how to implement the arithmetic functions and the Goodstein function on a Church-encoded *Ord* in lambda calculus.

abbreviation (*input*) *add_O* **where**
 $add_O \ n \ m \equiv (\lambda z \ s \ l. \ m \ (n \ z \ s \ l) \ s \ l)$

lemma *add_O*:
 $\langle add_O \ n \ m \rangle_O = add_O \ \langle n \rangle_O \ \langle m \rangle_O$
 $\langle proof \rangle$

abbreviation (*input*) *mul_O* **where**
 $mul_O \ n \ m \equiv (\lambda z \ s \ l. \ m \ z \ (\lambda m. \ n \ m \ s \ l) \ l)$

lemma *mul_O*:
 $\langle mul_O \ n \ m \rangle_O = mul_O \ \langle n \rangle_O \ \langle m \rangle_O$
 $\langle proof \rangle$

abbreviation (*input*) ω_O **where**
 $\omega_O \equiv (\lambda z \ s \ l. \ l \ (\lambda n. \ \langle n \rangle_N \ s \ z))$

lemma ω_O :
 $\langle \omega \rangle_O = \omega_O$
 $\langle proof \rangle$

abbreviation (*input*) *exp_{ω_O}* **where**
 $exp_{\omega_O} \ n \equiv (\lambda z \ s \ l. \ n \ s \ (\lambda x \ z. \ l \ (\lambda n. \ \langle n \rangle_N \ x \ z)) \ (\lambda f \ z. \ l \ (\lambda n. \ f \ n \ z)) \ z)$

lemma *exp_{ω_O}*:
 $\langle exp_{\omega_O} \ n \rangle_O = exp_{\omega_O} \ \langle n \rangle_O$
 $\langle proof \rangle$

abbreviation (*input*) *goodstein_O* **where**
 $goodstein_O \equiv (\lambda c \ n. \ n \ (\lambda x. \ x) \ (\lambda n \ m. \ n \ (m + 1)) \ (\lambda f \ m. \ f \ (m + 2) \ m) \ c)$

lemma *goodstein_O*:
 $goodstein_O \ c \ n = goodstein_O \ c \ \langle n \rangle_O$
 $\langle proof \rangle$

Note that modeling Church encodings with folds is still limited. For example, the meaningful expression $\langle n \rangle_N \ exp_{\omega_O} \ Z_O$ cannot be typed in Isabelle/HOL, as that would require rank-2 polymorphism.

7.1 Alternative: free theorems

The following is essentially the free theorem for Church-encoded *Ord* values.

lemma *freeOrd*:

assumes $\bigwedge n. h (s\ n) = s' (h\ n)$ **and** $\bigwedge f. h (l\ f) = l' (\lambda i. h (f\ i))$

shows $h (\langle n \rangle_O\ z\ s\ l) = \langle n \rangle_O (h\ z)\ s'\ l'$

<proof>

Each of the following proofs first states a naive definition of the corresponding function (which is proved correct by induction), from which we then derive the optimized version using the free theorem, by (conditional) rewriting (without induction).

lemma *add_O'*:

$\langle add_O\ n\ m \rangle_O = add_O\ \langle n \rangle_O\ \langle m \rangle_O$

<proof>

lemma *mul_O'*:

$\langle mul_O\ n\ m \rangle_O = mul_O\ \langle n \rangle_O\ \langle m \rangle_O$

<proof>

lemma *exp_O'*:

$\langle exp_O\ n \rangle_O = exp_O\ \langle n \rangle_O$

<proof>

end

References

- [1] J. C. Blanchette, M. Fleury, and D. Traytel. Formalization of nested multisets, hereditary multisets, and syntactic ordinals. *Archive of Formal Proofs*, Nov. 2016. http://isa-afp.org/entries/Nested_Multisets_Ordinals.html, Formal proof development.
- [2] E. A. Cichon. A short proof of two recently discovered independence results using recursion theoretic methods. *Proceedings of the American Mathematical Society*, 87:704–706, Apr. 1983. doi:10.2307/2043364.
- [3] R. L. Goodstein. On the restricted ordinal theorem. *Journal of Symbolic Logic*, 9:33–41, 1944. doi:10.2307/2268019.
- [4] J. Tromp. Binary lambda calculus. https://tromp.github.io/cl/Binary_lambda_calculus.html.
- [5] J. Tromp. Binary lambda calculus and combinatory logic. In C. S. Calude, editor, *Randomness And Complexity, from Leibniz To Chaitin*, pages 237–260. World Scientific Publishing Company, Oct. 2008.